Self-Adaptive Dynamic Decision Making Processes

K. Baclawski*, E. S. Chan†, D. Gawlick†, A. Ghoneimy†, K. C. Gross‡ and Z. H. Liu†
*College of Computer and Information Science
Northeastern University, Boston, MA USA
Email: kenb@ccs.neu.edu
†Oracle Corporation, Redwood City, CA USA
‡Oracle Corporation, San Diego, CA USA

***Abstract*—Decision making is important for many systems and is fundamental for situation awareness and information fusion. When a decision making process is confronted with new situations, goals and kinds of data, it must evolve and adapt. Highly optimized processes and efficient data structures generally have the disadvantage of having little flexibility or adaptability when confronted with new forms of data and new or changing goals. Consequently, optimized processes may only be locally optimal and may deteriorate over time. The normal approach to changing conditions is to manually reconfigure and even redevelop the system, which can be costly and time-consuming. In this article. we propose an architecture for the self-adaptation of decision making processes using flexible data structures and a process that monitors and adapts the decision making process. The objective is to have the ability to adapt both data schemas and decision making processes so that they can be both responsive and efficient.**

***Index Terms*—Decision support; situation awareness; expert systems; control; adaptive systems; flexible data**

I. Introduction

Software systems that must adapt to changing conditions normally must be reconfigured and even redeveloped by humans. Such adaptations can be costly and time-consuming. As a result, software systems are generally unresponsive to sudden changes in circumstances. To deal with this problem, systems have been developed that are self-adaptive, and this is an important research area. In this article, we propose an architecture for self-adaptation of software for decision making processes. Our architecture addresses not only the process of monitoring and adaptation but also the structure of the data that will be needed for restructuring and adapting the decision making process. The objective is to support both responsiveness and efficiency when conditions change.

An example of the need for adaptive decision making processes is the problem of controlling an aircraft. This involves many decisions that must be made, and there is a great deal of experience that is available to pilots and autopilots. However, circumstances can change very suddenly and dramatically. For example, a bird strike can cause complete loss of power, as in the case of US Airlines Flight 1549 [1], [2]. In this incident, the pilot, Sullenberger, had to develop a novel river landing procedure in a matter of only a few seconds. To develop the new procedure, Sullenberger had to combine a number of procedures based on his knowledge and experience. Sullenberger also had to verify that the combined procedure would satisfy a number of safety constraints, such as avoiding buildings and bridges. Modern autopilots are not sufficiently adaptable to be able to discover and verify such new procedures. This example is similar to many other emergency situations, such as nuclear power plant emergencies, and automobile driving emergencies. We will use this example to help explain our proposed self-adaptive decision making process.

Many kinds of self-adaptive system have been proposed. In Section II we survey related work in this area. Our proposed architecture offers additional features not currently available with existing approaches.

The framework we will use for decision making processes is Boyd's OODA loop [3]. The OODA loop has been expressed in terms of situation awareness in the Knowledge Intensive Data System (KIDS) framework developed in [4], and a formal ontological treatment has been developed in [5]. KIDS is based on the situation theory developed by Barwise and Perry [6]–[8], subsequently extended by Devlin [9], then formalized by the Situation Theory Ontology in [10]. We give an overview of these treatments in Section III. However, our discussion should apply to other dynamic decision making frameworks that are generally compatible with the OODA loop.

The specific aspects of dynamic decision making processes that are of concern for adaptation are the data and the functions that process the data. The evolution of the structure and the kinds of data being processed are especially difficult to adapt, and very few systems allow for evolution of this kind. Flexible and evolving data is the subject of Section IV.

The adaptation of the software components of a system has been the primary focus of self-adaptive systems research. We present our approach to software components for decision making in Section V. Our approach builds on previous work on self-controlling software.

The proposed architecture for adaptive, evolving decision making processes is presented in Section VI. The adaptation part of this architecture is dependent on decision making, which can be complex and involve many interdependent decisions that must be made. We end this article with a conclusion and outline of our plans for future work in Section VII.

II. Related Work

One of the earliest examples of an architecture for self-adaptation of software is the Self-Controlling Software Model (SCSM) developed in [11]. The SCSM controls the software system using control theory techniques and has three levels of control: feedback, adaptation and reconfiguration. The feedback level controls the software using a controller module. The adaptation level changes the controller in response to changing conditions. The reconfiguration level deals with dynamics that changes structurally in an unpredictable fashion. This level can change to a different design entirely that is selected from a library of alternatives that have already been trained. The reconfiguration level goes well beyond simply modifying various parameters as the adaptation level does; the reconfiguration level can redesign the structure of the software. The redesigned software is composed from modules that are checked for compatibility and correctness by using algebraic specifications. The SCSM addresses a number of issues that can occur when a system is self-reconfigurable such as generality, chattering, proactive reconfiguration, and efficiency. Although the SCSM specifies what the reconfiguration level should achieve, it does not have a process model for how the decisions are being made at the reconfiguration level, and it does not consider data adaptation. Our proposed architecture provides such a process model and addresses the adaptation of data.

For a survey and taxonomy of self-adaptive systems see [12]. Self-adaptive systems address the problem of the high cost of manually adapting software to changing conditions. Normally, manual adaptation of software involves reconfiguration and even redevelopment by humans. Such adaptations are not only costly but are also time-consuming. These problems are primarily due to the open-loop process of software development generally followed today. Accordingly, the response to these problems is to develop self-adaptive software that is a closed-loop system with a feedback loop aiming to adjust itself to changes during its operation [12]. Unfortunately, this can limit the possibilities for adaptation when open-loop interaction with the world is a fundamental aspect of the system. This is the case for the processes considered in this article that make decisions based on sensor data. Nevertheless, features of existing self-adaptive systems can be incorporated into the model proposed in this article when appropriate.

Self-adaptive systems can use machine learning as a mechanism for adaptation. Reinforcement learning (RL) is one class of machine learning algorithms for decision making that includes a mechanism for continuous adaptation [13]. Such algorithms attempt both to optimize performance (referred to as "exploitation") and to find new opportunities (referred to as "exploration"). The exploration feature of RL algorithms is a form of reconfiguration. The balance between exploitation and exploration is typically determined by a parameter that specifies the probability that an action is chosen at random rather than according to the currently learned behavior. While one can determine what the parameter should be when the system can be completely modeled analytically, for more complex systems that can have sudden changes in circumstances, one can only use various heuristics for selecting the parameter. The self-adaptive decision making model proposed in this article could be used not only to select the parameter but also to perform more dramatic reconfigurations such as substituting a previously learned RL model. Like the other self-adaptive models described in this section, RL algorithms do not consider data restructuring.

III. The KIDS Framework

The KIDS framework is a decision making and enactment process for situations that distinguishes different kinds of reasoning as well as different kinds of data in the decision making process [4], [14]–[16]. The kinds of reasoning in the KIDS framework correspond to the steps in the OODA Loop [3]. In the OODA loop, decision makers perform a series of steps repeatedly. The decision makers *Observe* the facts by capturing, fusing and filtering relevant data about the entities and environment. Then they use the information condensed from the facts to *Orient* within the unfolding situation. This step applies knowledge and personalization from sources such as prior experience, cultural traditions and genetic heritage. The results of this step are hypotheses that best explain the observations. Next the decision makers use the hypotheses to *Decide* on the directives. Then they *Act* on the directives to interact with the entities and environment and to test the hypothesis. The KIDS framework formalizes these four reasoning processes as instances of Classification, Assessment, Resolution and Enactment, respectively. Because of the large amounts of data that are now available, it is often necessary to perform relevance reasoning to select the data that form the situations in each reasoning process.

For airplane emergency example, the pilots had *observed* both the bird strike itself and the subsequent instrument readings. They needed to condense this information into an hypothesis about what the situation is. This is the *orient* step. The hypothesis was that the plan had suffered a loss of power in both engines. The next step was to *decide* what to do. The proper procedure was to verify their conclusion. Their *actions* included a number of instrument checks as well as attempting to restart the engines. At that point, they had established their situation awareness and notified controllers that they had an emergency.

Data is subclassed into four classes: *Fact*, *Perception*, *Hypothesis* and *Directive*. Each of these is the input of one reasoning function and the output of another. The reasoning functions are discussed in Section V below. The data produced by the last step in the process consists of action plans that affect the world, which is then observed to produce more facts, and the process then proceeds iteratively. This loop is shown in Figure 1. The reasoning actions in this loop are discussed in detail in Section V.

For airplane emergency example, the facts are the data about the airplane within its environment, such as its speed, altitude, heading, thrust, cabin pressure and temperature, to name a few. Most of this data is not relevant, of course. The perceptions are the instrument readings and direct observations by the pilots. Again, most of the instrument readings are not relevant. The hypothesis is that the airplane has lost all thrust. The directive is to verify the hypothesis using standard procedures.

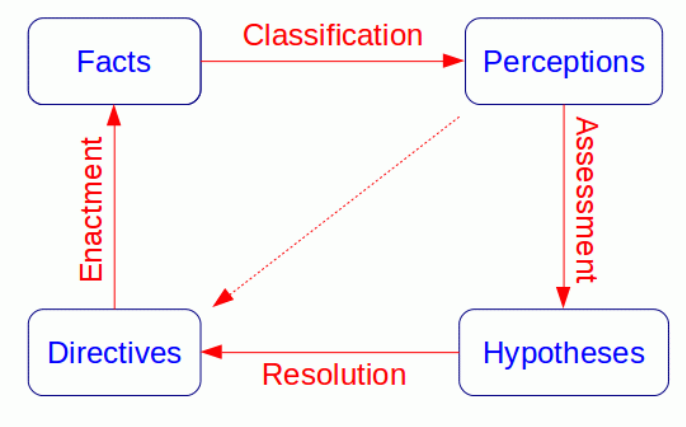

Fig. 1. Diagram of the dynamic decision cycle of the KIDS Framework

IV. Flexible Data

Traditional information storage systems, such as relational databases [17] and ontology-based systems, presume that the schema or ontology is developed prior to processing and storing the data. This assumption is not satisfied by many forms of data that are commonly needed in processes for situation awareness and decision making, such as human intelligence. Such data can be in a variety of forms ranging from unstructured documents, images and video to semi-structured documents, such as spreadsheets, spatial data and linked data [18]. This kind of data is unlikely to have a fixed schema or ontology. The best one can hope for is some minimal structure in the form of a schema or ontology that is continually changing. This kind of data is called Flexible Schema Data (FSD) [5].

There are many systems that already deal with FSD. Unstructured data such as text documents will often have a text index, while images and video will have associated metadata. Semi-structured data is represented in XML or JSON, and systems such as some relational databases and most NoSQL systems can process such data. However, there is little commonality among all of the formats and query languages for this great variety of unstructured and semi-structured data. Data platforms have incompatible capabilities and query languages. Most importantly, integration of such data is currently handled in an ad hoc manner by application programmers.

To deal with this problem, Liu and Gawlick [19] have proposed three principles to extend relational database platforms to manage FSD in the new extended relational systems; namely, principles for data storage, querying and indexing. These principles have been implemented in the Adaptive Schema Database (ASD) approach proposed in [20]. The ASD approach uses probabilistic curation to discover schemas and user feedback to improve the quality of the schemas. The ASD is responsible for decoupling the processes that are making use of the FSD data from the schemas used for data storage. This decoupling is important for allowing processes to make use of new kinds of data without the need for redeveloping the software that is responsible for the processes.

As we discussed in Section III, the KIDS ontology partitions data into four subclasses. Because modern sensors are inexpensive and produce data at very high rates, the volume of data in the *Fact* and *Perception* classes can be very large. Since much of this data is only semi-structured, and its structure evolves over time, it is necessary to use a flexible approach to the processing of the Fact data. The other subclasses of data in the KIDS ontology generally contain smaller amounts of data because they are outputs of reasoning processes, but in emergency situations, such as the airplane emergency example, the structure of all forms of data can evolve rapidly. In the airplane emergency, the pilot had to reason about data that is not normally part of a airline flight, such as the kinds and capacities of boats at the landing area.

V. Evolution of Functionality

The KIDS ontology also classifies reasoning processes and the activities performed by these processes. The four main classes representing reasoning processes are *Classification*, *Assessment*, *Resolution*, and *Enactment*. These reasoning processes are performed in a loop called the CARE loop. The CARE loop is the OODA loop in the KIDS framework. Each instance of one of these classes is a reasoning function that uses a situation as input and produces another situation as output. The overall dynamic decision making process is a loop as shown in Figure 1. Sensor data and other input data to the process are managed by the *Classification* functions. The overall output of the process is performed by the *Enactment* functions. An invocation of a reasoning function is an activity. Each activity includes the function being invoked, any parameters needed by the function, provenance information, as well as references to the input situation and the output situation. The classes of activities that correspond to the reasoning processes are *Classify*, *Assess*, *Resolve*, and *Enact*. The last of these activities, *Enact*, includes the goal that the CARE loop is intended to achieve. Other responsibilities of *Enact* activities include verification of the quality of the facts and determining whether there are any actionable perceptions that need to be created or updated [21].

In the airplane emergency example, verification of the situation was an important part of the initial stages of response to the emergency. The loss of an aircraft is not something that a pilot would consider without very strong evidence that there was no feasible alternative. Because of the dire

consequences of incorrect situation awareness, the verification process will normally involve more than one cycle through the OODA/CARE loop.

The situations used as input and produced as output by these activities are also classified. The requirements for the input situations and the characteristics of the outputs are also specifiable in the KIDS ontology. For more details, see [5].

The evolution of the CARE loop can be accomplished in two different ways. The first is to customize the reasoning functions without changing the functions being used. This is typically done by adjusting parameters. When a reasoning function is invoked, the parameters are included as part of the recording of the activity object. This is important for monitoring the CARE loop. The second approach is to select different reasoning functions. This strategy is used when the current reasoning functions are no longer adequate. This can happen for a variety of reasons, such as a changing environment, a changing goal, and new kinds of data.

In most airplane emergencies, the result of the emergency is to change the destination. This is more than just a case of adjusting some parameters, since a new flight plan is being developed and deployed. However, such an emergency does not normally require changing the reasoning processes that are used for executing the new flight plan. That was not the case for the Sullenberger airplane emergency example where a novel landing procedure had to be developed, deployed and executed.

VI. Proposed Architecture

Self-adaptation of a decision making process is driven by another decision making process, so it is natural for our proposed architecture to take advantage of the KIDS framework. To distinguish this new decision making process from the one that is being controlled and adapted, we call it the Adaptor Decision Making Process, or just the Adaptor. The facts that are used as input to the Adaptor can be classified into type classes: *DataStructure* and *ActivityStructure*. Instances of *DataStructure* are the kinds and sources of data that are being generated by the sensors. Such data are assumed to have been expressed using FSD, and are being managed by the ASD. When a new kind or source of data is available to the CARE loop, the ASD reports this to the Adaptor as an instance of *DataStructure*.

In the airplane emergency example, once the emergency was declared, the subsequent iterations of the Adaptor CARE loop had as their goal finding a feasible landing area. The sources of data for these iterations are aeronautical charts, especially the locations of airports. The situation of the airplane dictated that only the nearest airports or other landing areas were relevant. In the first iteration, only airports were considered. There were several nearby airports. This is not a new kind of data so now adaptation was required. However, when it was determined that no airport was a feasible destination, in later iterations, other sources of data about landing areas had to be used, and these did require a new kind of data. The eventual landing area on the Hudson River did not have any runways.

The instances of *ActivityStructure* are the parameters of the activities performed by the CARE loop. As noted in Section V above, an activity includes a variety of provenance information that, in principle, would allow the activity to be repeated. Only some of the activity information is used by the instances of *ActivityStructure*. As with any decision making process, relevance reasoning is performed to select those properties that are to be used for monitoring the performance of the CARE loop and that can be modified to improve performance. This relevance reasoning is performed statically.

In the airplane emergency example, the accident was thoroughly investigated by federal agencies who carefully considered all of the data on the airplane's data recorders, data from other sensors such as radars, and examinations of the wreckage. Provenance is important during such an investigation. Examples include the time when an event occurred, who is speaking during cockpit communications, what actions each pilot performed, etc.

The output of a CARE loop consists of directives that are the plans for enacting the decisions made by the CARE loop. For the Adaptor CARE loop, these directives are subclassified into two classes: *DataDirective* and *ActivityDirective*. Instances of *DataDirective* control the ASD by informing the ASD about which of the kinds and sources of data are relevant to the CARE loop. The Adaptor uses reasoning techniques to determine relevance, such as sensitivity analyses. Unlike the static relevance reasoning performed for *ActivityStructure*, this relevance reasoning is performed dynamically.

In the airplane emergency example, both new sources of data and new reasoning techniques had to be deployed. Once it was determined that no airport landing was possible, the pilots had to consider other possibilities. The requirements for landing a modern airliner are very stringent. The reasoning technique consists of applying the requirements to the geographic area within the range of the airplane. Even if a suitable landing area was close enough, the glide path had to avoid any obstructions. The pilots determined that there were two possible landing areas, both of which were water landing areas. In terms of the Adaptor CARE loop, these were hypotheses, each one being a landing situation. The Hudson River was one of the hypotheses.

The instances of *ActivityDirective* are modifications to be performed on the CARE loop. Such an instance can either be a change to the activity parameters or can be a restructuring operation in which different reasoning functions are used. One example of an activity directive is the tuning of the filtering associated with each reasoning process. Filtering selects the data that is specifically relevant to the situation that is input to a CARE step (such as selecting data that is nearby geographically). Note that this activity is not concerned

with the structure of the data, as that is the responsibility of the ASD. The architecture of the Adaptor is summarized in Figure 2.

In the airplane emergency example, the two possible water landings required substantial modifications to the landing procedures. Water landings are very risky and seldom successful. For obvious reasons, few pilots have actual experience with them, and even fewer have multiple experiences. Flight simulators could, in principle, provide such experience, but they do not currently have such a capability [22].

Another new activity required by the airplane emergency example is whether there would be sufficient boats to carry the passengers to shore after they evacuated the airplane. If there had been only one possible landing area, this would have been moot, but since there was a choice, the pilots had to use other information. This iteration of the Adaptor CARE loop used information that would not normally be part of any airline flight, and would not normally be available to the pilots. However, in this case, Sullenberger had personal knowledge about the boats that were available in each of the two landing situations. There were many more ferry boats on the Hudson and they had much greater capacity than the alternative. Accordingly, he made the decision to land on the Hudson.

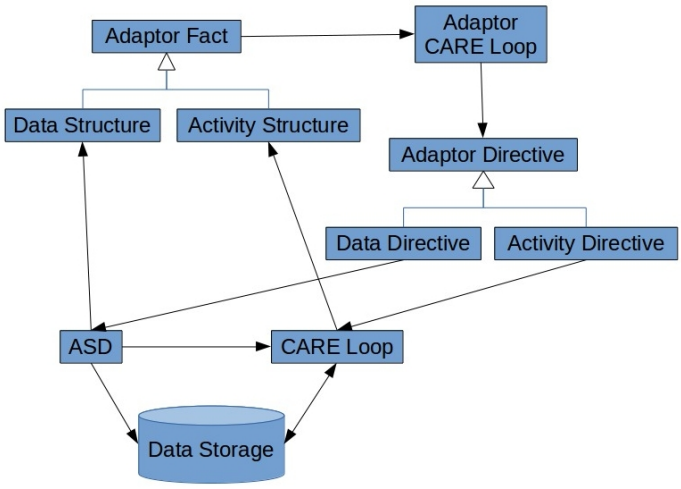

Fig. 2. Architecture Diagram for Self-Adaptive Dynamic Decision Making Processes

The Adaptor CARE loop runs at the same or somewhat lower frequency than the CARE loop being adapted. This allows the Adaptor to tune activities in the CARE loop in the same manner as a feedback control loop. However, restructuring is expected to be a relatively infrequent operation. The Adaptor architecture is compatible with the architecture of the SCSM in [11], which includes three levels of control, the highest of which is the reconfiguration loop.

There are four stages in the Adaptor CARE loop with the following responsibilities:

1) The Adaptor Classification stage performs data reduction on the Adaptor Facts. This generally would use statistical techniques for the ActivityStructure data. The DataStructure instances consist of schema metadata and may not require any data reduction, or possibly a decision tree can be used.
   In the airplane emergency example, both logical and probabilistic reasoning was used. To determine a feasible landing area, a number of logical constraints had to be satisfied as well as the likelihood that the landing area could be reached.
2) The Adaptor Assessment stage uses the output of the Adaptor Classification stage to determine whether to tune the parameters of the CARE loop or to restructure it. This can be done using a number of techniques. For the classified ActivityStructure information, such as the Multivariate State Estimation Technique using techniques such as Sequential Probability Ratio Tests or Bayesian Belief Networks [21]. For the DataStructure information, either a decision tree or Bayesian Belief Network can be used but there are many other techniques. The outputs of this stage can be expressed as hypotheses. This is the point where human intervention can most easily be included in the loop.
   In the airline emergency example, there were a series of hypotheses for the various iterations. The earliest iterations had as their hypothesis that the an emergency had occurred. These iterations relied primarily on logical deductions, but they would have benefited from other techniques that can deal with problems such as faulty or failed sensors. The later iterations were less logical and more probabilistic, as the pilots had to consider potential landing areas that are highly risky.
3) The Adaptor Resolution stage determines the directives for adapting the CARE loop being monitored. This can be done using decision trees or Bayesian Influence Networks. Directives that tune the CARE loop parameters can be computed using control theory techniques as discussed in [11]. Restructuring directives are more complex. When the *Fact* data structure of the CARE loop is changing, the restructuring decisions proceed forward through the CARE loop stages, by successively selecting compatible reasoning functions that can process the new input data structures. When the goal of the CARE loop has changed, the restructuring decisions proceed backward through the CARE loop stages, by successively selecting compatible reasoning functions that can produce the data structures needed by the new goal. Techniques such as Bayesian Belief Networks proceed in both directions through the stages.
   In the airline emergency example, the restructuring was extensive. The pilots had to “cobble” together a very non-standard landing procedure. Nevertheless, most of the components of the procedure were standard, including steps such as banking, trimming, and managing the control surfaces. The components of the new procedure

have to be selected, checked for compatibility with one another and then combined to form the overall procedure.

As noted in [5], the Assessment and Resolution stages can be combined if there is no reason for explicitly generating hypotheses. This could be done, for example, if the Adaptor is not being monitored by a human or other process.

4) The Adaptor Enactment stage modifies the CARE loop and ASD as shown in Figure 2. Modifying reasoning function parameters is relatively easy to enact. Restructuring is more complex. If the new configuration is a already known, then one can restructure by simply installing the new functionality. However, if the configuration is new, then a training period will be necessary. This will use machine learning techniques and may require human supervision.

   In the airplane emergency example, once the pilots had constructed the new procedure, they carried it out. Needless to say, no training period was possible. It was not quite as successful as Sullenberger had hoped, since the automated systems of the airplane did not allow him to perform the maneuvers in exactly the way he wanted. While the pilots had a good understanding of the new procedure and were executing it, the airplane's systems were not restructured. Indeed, alarms were sounding continually until the plane stopped moving.

Since the Adaptor is structured as a CARE loop, it will include the same features as any CARE loop. An important aspect of a CARE loop is the recording of provenance information. This part of the KIDS ontology uses the Provenance Interchange Ontology [5], [23]. Provenance information includes not only the "change tracking" information, but also the rationales for the modifications made by the Adaptor.

VII. Conclusion and Future Work

We have proposed an architecture for a self-adaptive dynamic decision making process. This architecture makes use of known techniques for flexible data, self-adaptation and decision making, but organizes them in a novel manner. While the intention was to focus on the specific scenario of self-adaptation of a decision making process, the same techniques could be employed for self-adaptation of more general processes. In particular, the KIDS framework and ontology has proven to be a versatile approach for formalizing applications in a large variety of domains.

Acknowledgments

We wish to acknowledge the continuing support of Oracle to the development of the KIDS framework and ontology.

References

[1] "US Airways Flight 1549," 2009 [Online], Available: https://en.wikipedia.org/wiki/US_Airways_Flight_1549.

[2] C. Sullenberger and J. Zaslow, *Sully*. William Morrow, 2009.

[3] J. Boyd, "Destruction and creation," U.S. Army Command and General Staff College, Tech. Rep., September 3 1976 [Online], Available: http://www.goalsys.com/books/documents/DESTRUCTION_AND_CREATION.pdf.

[4] D. Gawlick, E. Chan, A. Ghoneimy, and Z. Liu, "Mastering situation awareness: The next big challenge?" *SIGMOD Record*, vol. 44, no. 3, pp. 19–24, 2015.

[5] K. Baclawski, E. Chan, D. Gawlick, A. Ghoneimy, K. Gross, Z. Liu, and X. Zhang, "Framework for ontology-driven decision making," *Applied Ontology*, 2017, to appear.

[6] J. Barwise, "Scenes and other situations," *J. Philosophy*, vol. 77, pp. 369–397, 1981.

[7] J. Barwise and J. Perry, *Situations and Attitudes*. Cambridge, MA: MIT Press, 1983.

[8] J. Barwise, *The Situation In Logic*. Menlo Park, CA: CSLI/SRI International, 1989, vol. 17.

[9] K. Devlin, *Logic and Information*. Cambridge, U.K.: Cambridge University Press, 1991.

[10] C. Matheus, M. Kokar, and K. Baclawski, "A core ontology for situation awareness," in Proc. Sixth Intern. Conf. on Information Fusion FUSION'03, July 2003, pp. 545–552.

[11] M. Kokar, K. Baclawski, and Y. Eracar, "Control theory-based foundations of self-controlling software," *IEEE Intelligent Systems and their Applications*, vol. 14, no. 3, pp. 37–45, 1999.

[12] M. Salehie and L. Tahvildari, "Self-adaptive software: Landscape and research challenges," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 4, no. 2, May 2009, doi:10.1145/1516533.1516538.

[13] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*. Cambridge, Massachusetts: The MIT Press, 1998.

[14] Z. Liu, A. Behrend, E. Chan, D. Gawlick, and A. Ghoneimy, "KIDS - A model for developing evolutionary database applications," in Proceedings of the International Conference on Data Technologies and Applications (DATA 2012), Rome, Italy, 25-27 July 2012, pp. 129–134.

[15] E. Chan, A. Behrend, D. Gawlick, A. Ghoneimy, and Z. Liu, "Towards a synergistic model for managing data, knowledge, processes, and social interaction," in Society for Design and Process Science (SDPS), 2012.

[16] E. Chan, D. Gawlick, A. Ghoneimy, and Z. Liu, "Situation aware computing for big data," in Workshop on Semantics for Big Data on the Internet of Things (SemBIoT 2014), 2014 IEEE International Conference on Big Data, Washington DC, October 27–30 2014.

[17] E. Codd, "A relational model of data for large shared data banks," *Commun. ACM*, vol. 13, no. 6, pp. 377–387, 1970.

[18] C. Bizer, T. Heath, and T. Berners-Lee, "Linked data–the story so far," *International Journal on Semantic Web and Information Systems*, vol. 5, no. 3, pp. 1–22, 2009, doi:10.4018/jswis.2009081901. ISSN 1552-6283.

[19] Z. Liu and D. Gawlick, "Management of flexible schema data in RDBMSs via naturally open set-oriented query language (NoSQL)," 2016. Oracle.

[20] W. Spoth, B. Arab, E. Chan, D. Gawlich, A. Ghoneimy, B. Glavic, B. Hammerschmidt, O. Kennedy, S. Lee, Z. Liu, X. Niu, and Y. Yan, "Adaptive schema databases," in 8th Biennial Conference on Innovative Data Systems Research (CIDR '17), Chaminade, California USA, January 8–11, 2017.

[21] K. Gross, K. Baclawski, E. Chan, D. Gawlick, A. Ghoneimy, and Z. Liu, "KIDS supervisory control loop with MSET prognostics for human-in-the-loop decision support and control applications," in CogSIMA 2017, 2017, submitted.

[22] "Aviation StackExchange," 2014 [Online], Available: http://aviation.stackexchange.com/questions/7835/can-water-landing-be-simulated.

[23] "PROV Ontology (PROV-O)," April 30 2013 [Online], Available: http://www.w3.org/TR/2013/REC-prov-o-20130430/.