A Taxonomy of Artificial Intelligence Approaches for Adaptive Distributed Real-time Embedded Systems

Jeremy Davis, Joe Hoffert, Erik Vanlandingham
Department of Computer Science
Indiana Wesleyan University
Marion, IN 46952
Email: jeremydavis519@gmail.com, joe.hoffert@indwes.edu, vanlandingham1138@gmail.com

***Abstract*—Distributed real-time embedded (DRE) software systems such as are used to manage critical large-scale infrastructure are important systems to target for increased functionality and resiliency. DRE systems that can adapt to changes in the environment and/or changes in available resources are more robust to unexpected changes and extend both the systems' utility and lifespan. Artificial intelligence techniques are used for adaptation of software systems in general. However, they must meet certain requirements to be appropriate for use with DRE systems.**

Any artificial intelligence (AI) technique used in an adaptive DRE system should produce consistent results across a distributed system, operate in bounded time, make decisions autonomously, and gracefully handle and learn from previously unencountered environments. This paper surveys a variety of AI techniques, providing a brief overview of each method, evaluating how each technique fits the requirements of an adaptive DRE system, and recognizing the gaps in each technique's ability to meet all of these requirements. Our results show that there is not one single AI technique in our survey that is a perfect fit for adaptive DRE systems, although some techniques address more of these requirements than others.

I. Introduction

Distributed real-time embedded (DRE) software systems are used to manage critical large-scale infrastructure across a wide range of platforms and domains. Such systems include air traffic management, regional and national power grids, shipboard computing environments for military platforms, and disaster recovery operations [11]. The importance of these systems motivates the need for flexibility so that the systems can respond appropriately in situations not previously planned. However, traditionally these types of systems have been developed for rigidly controlled environments and resource allocations determined *a priori*.

DRE systems that can adapt to changes in the environment and/or changes in available resources are more robust to unexpected changes and extend both the usefulness of the systems and their lifespans. One approach to provide adaptability for DRE systems is to incorporate artificial intelligence (AI) techniques that can respond to changes in the environment and in resource allocation. The benefits of this adaptability are two-fold: (1) managing expected environment and resource changes can be simplified and (2) appropriate responses can be generated for unexpected changes.

Many AI techniques have been developed over the past several decades. However, not all of these techniques are equally appropriate for use with adaptive DRE systems. Some AI techniques do not address critical properties that DRE systems must have in order to provide the functionality needed.

This paper provides insight into adaptive DRE systems in the following ways. (1) It provides an enumeration and description of critical properties of adaptive DRE systems. (2) It surveys and briefly describes common AI techniques. (3) It taxonomizes the AI techniques based on the critical properties of adaptive DRE systems. (4) It provides gap analysis of the AI techniques to explore where additional research is needed when incorporating AI techniques into adaptive DRE sytems.

II. Motivating Example - Power Grid Stability in Unexpected Circumstances

Software control of regional and national power grids — referred to as smart grids [18][22] — is increasing in popularity due to the complexity of grid management and the flexibility and responsiveness that software systems provide. In particular, adaptive DRE systems support smart grids well because of (1) the distributed nature of disseminating power, (2) the timeliness requirements of addressing changes in the system, and (3) the embedded nature of a widely diverse system which precludes regular direct human control and intervention. This section describes the motivating example of an electrical smart grid that is controlled by adaptive DRE software. This example is useful when evaluating various AI techniques for use with an adaptive DRE system.

In 2003, 50 million people lost power in the northeast U.S. and southeast Canada [15]. The problem can be traced back to a single power line that went down in northern Ohio. When a power line goes down the demand for power is re-routed through other lines. The transformers and the power lines themselves can be overloaded so that they shut down. Power lines can soften due to the heat of the high current since additional power is being re-routed through them. Once one power line or transformer is shut down the needed power is re-routed yet again and there can be a cascading shutdown effect. Without timely intervention and flexible response to the initial overload conditions, widespread loss of power can occur causing inconvenience, damage, and even loss of life.

Power grid service is critical to the healthy functioning of regional and national infrastructure. Loss of power can mean loss of life both directly (*e.g.*, medical systems used to maintain patient health) and indirectly (*e.g.*, traffic light outages resulting in deaths from crashes). Moreover, loss of power can be very costly for businesses and services that rely upon that power [3].

While disruptions can occur in the environment (*e.g.*, substation outages due to inclement weather) and in the system itself (*e.g.*, upgrades and maintenance) adapting to changes and maintaining acceptable service is crucial for modern day power grids. Power grid systems leverage the properties of adaptive DRE systems to manage changes in the environment and/or the system. These adaptive DRE properties are enumerated as follows:

Adaptive changes should be consistent across the distributed power grid. Otherwise the power grid will be in an inconsistent and potentially unstable state. An inconsistent view of the system can cause unneeded and potentially dangerous local changes. An

unnecessary shutdown in one part of the grid could cascade throughout the entire grid with catastrophic consequences.

Adaptive changes should be done in a bounded amount of time. Otherwise the adaptation that is carried out will be too late to be effective. An appropriate change that is implemented too late becomes an inappropriate change when timeliness is important. Untimely changes could worsen the status of the power grid by creating instability and initiating changes that are no longer appropriate because the overall state of the grid has changed.

Adaptive changes should be made without human input. Otherwise the changes might not occur in a timely manner. Some adaptive changes need to occur faster than humans can react (*e.g.*, shutting down a substation before it is overloaded). In addition, human interaction can introduce human error and worsen the state of the power grid. Moreover, the size, complexity, and wide geographical deployment of the grid makes human intervention infeasible.

Adaptive changes should be robust to unforeseen perturbations. Otherwise, undesirable performance or behavior might result from these perturbations. Circumstances that weren't previously predicted can occur with a system as large and complex as a power grid and any adaptive change should gracefully handle these circumstances even if these circumstances have not been accounted for *a priori*.

Adaptation should incorporate unforeseen perturbations into an overall strategy. The power grid's resiliency will be lessened if it can not learn from previous changes. So that the power grid will be more resilient, it should incorporate any unforeseen circumstances into its adaptation strategy. Once the initially unforeseen circumstances are handled the power grid should adapt to anticipate these circumstances should they arise again. Therefore, the next time these circumstances are encountered the grid can better respond.

III. Taxonomy for Adaptive DRE Systems

This section presents the taxonomy for adaptive DRE systems. For any approach used in providing adaptation for a DRE system the following areas should be considered.

1) **Distributed.** The adaptive approach used should support a distributed environment. One tactic for supporting the distributed property is to provide deterministic behavior. This behavior can then be consistently replicated on all the computational nodes or machines utilized in this system. With this approach the behavior for any one computational node is known *a priori* and will be the same across all nodes.
2) **Real-time.** The adaptive approach used should support real-time timeliness requirements. At a minimum any approach used for adaptation should provide bounded-time complexity (*i.e.*, the worst-case time is known *a priori*). This timeliness property allows for the analysis and design of the system such that allocation and deployment of resources will meet the needed timeliness requirements. Ideally, the time complexity will be constant time to ease analysis and development.
3) **Embedded.** The adaptive approach used should support an embedded environment. An embedded computer system is developed in such a way that the user is not consciously aware that they are interacting with a computer system. It presents an interface that is familiar and common for the domain of the user. For example, an embedded computer system can be used in a vending machine that dispenses drinks. The user interfaces with the machine by inserting money and making a selection by pushing buttons. The computer system decides if the appropriate money has been deposited as well as determining if the selection is valid (including checking that a normally valid selection is still available).

   Embedded computer systems are typically constrained in the resources available (*e.g.*, memory, computational power) since the system must conform to particular space and footprint requirements which limit space for the computational resources. Embedded computer systems also typically cannot rely on expert human intervention. The user interacting with the system will generally not be a computer system expert. Therefore, the computer system needs to be autonomous in its execution and decision-making.
4) **Robust to new inputs.** Adaptive DRE systems need to be robust in handling inputs that have not been previously encountered. A critical property for adaptive DRE systems is gracefully handling unexpected situations. The adaptive approach used should be able to provide reasonable output in these cases even if the output is less than ideal.
5) **Autodidactic.** The adaptive approach used should support being updated with new information as it becomes available. As the adaptive DRE system runs it will encounter new information relevant to its adaptation. For example, if a system is designed to adapt to a new environment configuration, there will be environment configurations the adaptive approach has not encountered previously.

   The adaptive approach should handle any new information appropriately (as referenced in the previous property) but additionally should leverage this new information to prepare for additional new configurations. For example, supervised machine learning approaches should use this new information as additional training data. Moreover, when the learning approach is leveraging the new information as part of its training data the other relevant properties (*i.e.*, distributed, real-time, embedded, and robust) must continue to be supported.

IV. AI Approaches

This section provides a brief overview of various AI approaches considered, highlighting distinguishing aspects of each approach. This overview is leveraged in Section V where the paper taxonomizes the approaches based on the categories described in Section III.

• **Reinforcement Learning.** Reinforcement learning [12][19] is an approach where general, broad-based negative and positive reinforcement is used to guide learning. For example, a program or agent developed to learn strategies for winning a game would receive feedback when the game was won (*i.e.*, positive reinforcement) or lost (*i.e.*, negative reinforcement). From this general reinforcement, the program would develop strategies to determine what moves to make and not make. These strategies are generally developed based on states and actions from and to those states. Several designs are used for developing strategies including:

1) a utility-based agent: which learns a utility function on the different states and uses this function to select an action that maximizes the expected outcome;
2) a Q-learning agent: which learns a utility function on the different actions (*a.k.a.* Q-function) and selects the action with the highest utility; and
3) a reflex agent: which learns a policy mapping directly from states to actions.

Each of these designs provides advantages and disadvantages. For example, a utility-based agent must be aware of the envi-

ronment and the related states because it needs to know which states the actions will lead to. On the other hand, a Q-learning agent can compare expected utilities for available choices without knowing the outcomes. However, Q-learning agents cannot look ahead, which can restrict the ability to learn.

• **Expert Systems.** Expert systems [19] are used to leverage knowledge from domain experts. An expert system can be thought of as a surrogate for human experts. This approach is motivated by the lack of expert availability and coordination concerns (*e.g.*, logistics of gathering all relevant experts in one place at one time). Experts are solicited for general rules that they use to make decisions within a particular domain (*e.g.*, medical diagnosis, network configuration). These rules are entered and supported by the expert system along with inferences that can be made between the rules and the domain information available.

An early example of using expert systems is found in the DENDRAL program [2]. This program was developed to infer molecular structure from the results of a mass spectrometer. DENDRAL leveraged the knowledge of analytical chemists to shrink the intractable search space to a manageable size. The success of DENDRAL initiated the practice of mapping a general form to an efficient special form [6] encapsulated in expert systems.

• **Decision Trees.** A decision tree utilizes the tree data structure containing vertices and edges (a.k.a. branches). Non-leaf vertices correspond to decisions made regarding attributes in the solution space. Leaf vertices result in a classification made based on the attributes. A decision tree is typically a binary tree where the two branches leaving a node correspond to the presence or absence respectively of the attribute for that vertex.

Each node references the attribute that yields the greatest divide between the data samples being classified [19]. This design produces the shortest and smallest tree possible while maintaining accuracy by looking for features that best split the data as completely as possible. Non-binary attributes (*i.e.*, attributes with more than two possible values) can be referenced in more than one node based on the value of the attribute. For example, one node could split the data samples based on the value of an attribute being less than 10. Another node could split the data samples based on the same attribute being greater than 100. A leaf node determines the classification for the input data.

• **Random Decision Forests.** Random Decision Forests are a composition of Decision Trees where attributes are randomly selected to generate a variety of Decision Trees. The randomness of attribute selection results in different classifications and the multitude of trees provides a more generalized coverage of attributes as opposed to any single Decision Tree. After the forest is created, each tree will give a classification for the input data. The forest will select the most common classification returned by the composition of trees [14]. For the remainder of this paper we have Random Decision Forests subsume the AI technique of Decision Trees since they are more generalized w.r.t. adaptive DRE systems.

There are several strategies to introduce randomness in a Random Forest. One strategy is to select random best-split characteristics attributes from the input data [1]. The second method strategy is to randomly select which best-split features the tree will split on [5]. A third method randomly selects subsets of the input data to achieve 100% accuracy in classification on training data and increased accuracy in classification of unknown data [9]. Lin and Yongho [14] also mention a fourth method called Extreme Randomness which combines the randomness in attribute selection and in splitting features.

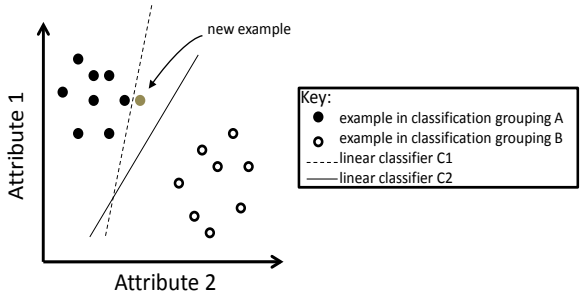

Fig. 1. **Support Vector Machine Example**

• **Support Vector Machines.** Support Vector machines (SVMs) are designed to find the greatest equal divide between classification axes for groups of input data. The SVM uses training data to build a model of data classification and obtain the greatest divide [10]. The training values are flagged with a classification schema so that when new data points are entered the SVM will use the separation to determine which schema-classification is appropriate for the new point. Finding the greatest separation will result in increased accuracy in classification for unknown input data.

Fig. 1 shows group 1 (denoted by filled circles), group 2 (denoted by open circles), and an unknown input in gray. The circles in black are training examples, and the solid line is the greatest separation between the groups that the SVM will use to classify to which group the new entry belongs. An SVM would select the solid line for classification, which maximizes the likelihood of appropriate classification for any new data. When new data becomes available (as denoted by the gray circle) this SVM classifies the gray circle into group 2. The dashed line represents a possible classification that accurately separates the existing data but does not perform as well with new data.

• **Artificial Neural Networks.** Artificial Neural Networks (ANNs) are based on a biological model of the human brain. An ANN has the useful ability to find patterns in empirical data to generalize its behavior to similar, but not previously encountered, data [16]. Each node in the network, called a *neuron*, outputs a nonlinear transformation of a weighted sum of its inputs — that is, $f(\sum w_i x_i)$ for some function f (known as the activation function), inputs x_i, and associated weights w_i.

These neurons are typically arranged in discrete layers, namely the input layer, zero or more hidden layers, and the output layer. Nodes in the input layer receive data from the environment, and nodes in the output layer produce the network's learned response to the given input. The hidden layers lie between the input and output layers and are "hidden" in that their states cannot be known based on the input and output alone. In general, each layer produces input for the neurons in the next layer, although some networks' structures (called recurrent) [16] send some output from one layer to an earlier layer or to nodes in the same layer.

• **Bayesian Networks.** A Bayesian Network (BN) is a structured representation of the attributes of a system and their interconnected probabilities which represent the system as a coherent whole [4]. The network establishes this structure by linking its nodes together in a directed acyclic graph of causal relationships, where for nodes *A* and *B*, $A \rightarrow B$ means, roughly, that the value of node *A* is one cause of the value of node *B*. This structure can then be used to infer the most likely state of any given node given the state of any or all of the other nodes in the network. For instance, if *A* represents contact with a diseased person and *B* represents contraction of the disease, then *A* causes *B*, and there is some

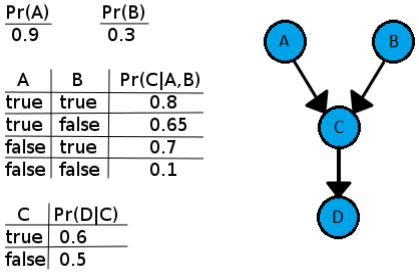

Fig. 2. **Bayesian Network Example**

probability, written as $Pr(B \mid A)$, that, given that A occurs, B also occurs. This probability is stored in the Bayesian Network in the data structure for node B, and the probability of A occurring in the first place, $Pr(A)$, is stored in node A.

Figure 2 provides an example BN where each variable can be either true or false. In general, it is possible for a node to have any number of possible states. Node C's probability table contains four rows with the understanding that the four probabilities that C is false are easily calculable from the probabilities that C is true. Likewise, node D's probability table is also listed based on the value of node C.

Because it is generally easier for a BN to learn the best values for its probability tables than to learn an appropriate network topology, it is common to assume some canonical network structure when learning BNs from data, such as the particularly common *naïve Bayes* [4]. In a naïve BN, there is one node C, typically referred to as the class variable, and one or more nodes A_i, referred to as attributes, such that $C \rightarrow A_i$. This structure, and the slightly more complex *tree-augmented naïve Bayes*, perform surprisingly well for classification problems in general [7][17].

V. Classification of AI Approaches

This section classifies or "taxonomizes" AI approaches based on the needs of adaptive DRE systems as outlined in section III and as summarized in Table V. In particular, each AI approach surveyed is classified on the 5 properties needed for adaptive DRE systems, namely (1) supporting a distributed environment, (2) supporting real-time requirements, (3) supporting an embedded environment, (4) robustly handling new data, and (5) incorporating new data into the approach as it becomes available while the system is running.

• **Artificial Neural Networks.** An ANN is generally a good fit for an adaptive DRE system, but it suffers from difficulty with learning over time.

Distributed. After training, an ANN's output is deterministic as it depends only on the input, the network's static structure, the weights that have been learned, and the neurons' activation functions. Thus, identical ANNs distributed across a DRE system will all reach the same conclusion given the same input.

Real-time. An ANN is ideal for a real-time system because the time it takes to process any set of input data is bounded by the number of connections between the neurons, and that number is known at compile-time. As a result, the computation time of a given ANN is constant.

Embedded. Because an ANN maps each set of inputs directly to a set of actionable outputs, even for inputs that it has not yet encountered, it is ideal for making decisions without human intervention.

Robust to new inputs. Regardless of the training set, an ANN can produce output for every input set. Moreover, it interpolates between the training outputs when used on inputs outside of the training set, which can improve accuracy if the output values can be placed in a meaningful order (*e.g.* low, medium, and high). In general, an ANN can be used reliably on both known and previously unencountered inputs for a classification given a set of attributes, such as the attributes that determine the communication protocol that should be used to maintain a given quality of service [10].

Autodidactic. Unfortunately, an ANN does not easily learn from experience. The correct output must be known before the ANN can learn, which implies either a more accurate AI technique employed alongside the ANN, human intervention, or a problem whose solution becomes obvious in retrospect. Then the ANN must be trained on the previous data set in addition to the new information that it is to learn. This training may conceivably be done on a separate processor in order to preserve the real-time requirement for using the network, but the time required for training is unbounded. The training is, however, deterministic, which is amenable for adaptivity in a distributed system.

• **Bayesian Networks.** The general definition of a BN offers few guarantees about its timeliness or how it can learn over time. If a network changes its structure over time (*e.g.*, for the purpose of learning), then the time required for evaluating the network is potentially unbounded. The rest of this section assumes a static structure to guarantee the possibility of real-time operation with the trade-off of potentially losing accuracy.

Distributed. A BN, although it deals with probabilities, is deterministic because it deterministically calculates results. Any Bayesian Network provides the same conclusion given the same probabilities and network structure.

Real-time. Every prediction by a BN can be bounded by the product of the number of nodes in the network, the number of possible states for each node, and the number of connections between them. An explicit bounded-time mathematical formula can calculate any probability in the network, using every connection and every possible state of a node no more than once.

Embedded. Since a BN is used to determine the most likely value of a node given all available information, it can make decisions with high accuracy without human intervention, as embedded software systems require.

Robust to new inputs. A BN deals directly with probabilities, so it can be guaranteed to produce for any question the most likely answer given everything that it has learned. However, the conclusion of a Bayesian Network with discrete values for its nodes — which are typically used for the purposes of timely computation and ease of understanding — can be skewed by its lack of support for interpolation. For instance, if a BN is designed to classify a risk as either low or high, then it will never indicate any level of risk between those two extremes.

Autodidactic. Some BNs can learn at run-time easily and in constant time by storing frequency counts rather than probabilities. In that case, each node contains a frequency for each of its possible states, and each connection holds a frequency for every combination of the states of its endpoints. Increasing each frequency by the associated event's new probability of having just happened (1 or 0 if the node's current state is known) will move the initial probability of every state of a node toward that new probability. The proof is fairly simple but beyond the scope of this paper. Learning thus runs in bounded time, is deterministic, and needs no human intervention.

• **Expert Systems.** While expert systems support some of the properties for adaptive DRE systems, multiple factors make expert

AI Method	Distributed	Real-Time	Embedded	Robust	Autodidactic
Artificial Neural Network	✓	✓	✓	✓	✗
Bayesian Network (dynamic structure)	✓	✗	✓	✗	✗
Bayesian Network (static structure)	✓	✓	✓	✗	✓
Expert System	✓	✗	✗	✗	✓
Reinforcement Learning	✓	✗	✓	✗	✗
Random Forest	✓	✓	✓	✓	✗
Support Vector Machine	✓	✓	✓	✓	✗

TABLE I
SUMMARY OF AI TECHNIQUES

systems less than ideal approaches in these contexts.

Distributed. Expert systems are appropriate for use in distributed environments since the results of using an expert system are deterministic. Expert systems with identical facts, rules, and inference engines can be deployed on computing nodes in the distributed system so that all nodes will generate the same advice.

Real-time. Expert systems typically utilize facts, rules, and an inference engine to provide solutions to problems based on the knowledge of human experts. The inference engine uses the facts and rules to infer knowledge which can also create new facts and initiate the use of additional rules. The inference engines creates this lattice or web of information that interconnects rules and facts and changes dynamically.

However, the selection of relevant rules and facts within the inference engine depends upon the problem being addressed. Some rules and facts will not apply for particular problems. Accordingly, searching the rules and facts in general is not optimized for a specific problem and the time complexity — while not strictly unbounded — is intractable in practice [21].

Embedded. Expert systems are typically used for decision support and provide guidance for human operators rather than being autonomous and implementing the advice given. This property makes expert systems not well suited for embedded environments.

Robust to new inputs. Expert systems do not support interpolation or extrapolation of results. Inferences are made from the facts and the rules. However, unless adaptation decisions involve specific facts and rules an expert system will not be able to provide suitable guidance. Therefore, an expert system is not robust to handling inputs not previously encountered.

Autodidactic. Since expert systems are based on facts and rules they are amenable to adding new facts and rules as they become available. Adding a rule or fact for the inference engine is straightforward (*e.g.*, stating a fact for a Prolog knowledge base) and can be done in constant time. The inference engine will utilize any new fact or rule as soon as it has been added. However, the addition of new rules, if the number of new rules is unbounded, leads to truly unbounded time complexity in the generation of inferences.

• **Reinforcement Learning.** While reinforcement learning has good properties in general of exploring a solution space, its utilization of pseudo-randomness and probabilities makes it ill-suited for adaptive DRE systems.

Distributed. Reinforcement learning can be distributed across nodes in a distributed system. However, a key aspect of reinforcement learning is the use of stochastic or probabilistic transitions. To reach a consensus where pseudo-randomness is not consistent across nodes would require a consensus protocol such as that outlined in the Byzantine Generals Problem [13]. To address this problem each node could use the same seed(s) to generate pseudo-random values so that each node will make the same transitions at the same time.

Real-time. Due to using probabilities in transition functions, the time complexity of reinforcement learning is unbounded. Potentially, there are cycles in the possible transitions so that transitions could traverse cycles an unbounded number of times. Therefore, reinforcement learning does not satisfy the bounded timeliness property of DRE systems.

Embedded. When developing reinforcement learning, feedback is needed to determine ultimate success or failure. However, after reinforcement learning has been adequately trained, there is no intervention needed and the reinforcement learning can be used without human intervention.

Robust to new inputs. Reinforcement learning is designed to determine a path (via transitions) from a start state to an end goal. If the goal is changed (via new inputs) then reinforcement learning will need to relearn its transitions. If the goal changes then the reinforcement learning needs to change as well. Therefore, reinforcement learning is not robust to new inputs.

Autodidactic. As noted previously, the learning time for reinforcement learning is unbounded. Therefore, reinforcement learning does not fulfill the autodidactic property for adaptive DRE systems.

• **Random Forests.** Random Forests can be adapted to fulfill most of the requirements of a DRE system.

Distributed. There are two possible ways of achieving determinism in a Random Forest. While pseudo-randomness is used to determine the trees in the forest, this pseudo-randomness can be limited to each node in the distributed system. The same pseudo-random value can be deterministically achieved using the same seed on each node. Alternatively, if the pseudo-randomness is non-deterministic (*i.e.*, each node potentially uses a different seed) across all nodes, then consensus needs to be reached across the nodes using a protocol similar to the one outlined in the Byzantine Generals Problem [13]. The latter solution provides unbounded time complexity due to the potentially unbounded number of trees and tree depth. Therefore, Random Forests should limit pseudo-randomness to within a node and not across nodes.

Real-time. The timeliness of the Random Forest is dependent on the average depth n of the trees and the number of trees m in the forest, *i.e.*, O(m log n) [14]. It is possible to bound time complexity further with depth limitations, tree pruning, and the complex method of feature splitting [8] to address timeliness concerns. However, this bounding has implications for accuracy.

Embedded. Random Forests are decision making systems capable of reaching a classification without the interaction of a human, fulfilling the requirement of embedded systems.

Robust to new inputs. A Random Forest's ability to generalize is determined by the number of trees in the forest. As the number

of trees increases, the generalization increases as long as each new tree is sufficiently different from all the other trees in the forest [8].

Autodidactic. For a Random Forest to incorporate new data, new classification trees of non-predetermined depth must be added to the forest [14]. Time complexity will become unbounded without depth limitations, limitations on the number of trees in the forest, and tree pruning. If the forest is limited in either the number of trees that can be added or the depth and breadth of each individual tree (which is needed to address timeliness concerns), accuracy can be lost. Therefore, Random Forests do not address the autodidactic needs of adaptive DRE systems.

• **Support Vector Machines (SVMs).** SVMs are generally useful for adaptive DRE systems. They are particularly useful for processing inputs at run-time that have never been seen before, which emphasizes their adaptability and accuracy under such conditions [11].

Distributed. SVMs are ideal for distribution due to the deterministic nature of their decision making process.

Real-time. Training of an SVM is unbounded. However, after the initial training is completed SVMs classify in constant time.

Embedded. SVMs are decision making systems — their classifications are done without the aid of human intervention allowing them to be implemented in an embedded system.

Robust to new inputs. SVMs support classification for new inputs. Moreover, in certain situations the accuracy of an SVM w.r.t. new inputs has been shown to be better than other AI methodologies such as ANNs [11].

Autodidactic. In order for an SVM to incorporate new classifications into the training set, the SVM has to be retrained. While an SVM is robust to data not included in training, the accuracy of an SVM is limited if it is not retrained with the new data. With an unbounded time complexity for training, SVMs do not support the autodidactic property.

VI. Analyzing Gaps

Per analysis of the AI techniques for adaptive DRE systems, no single technique fully addresses all five areas of concern. Additionally, no AI technique that is described in this paper sufficiently and necessarily addresses the autodidactic property of adaptive DRE systems aside from Expert Systems, which have several other deficiencies. Some of the techniques start to address the autodidactic property if certain constraints are applied (*e.g.*, restricting Bayesian Networks from dynamically modifying the network topology). There are AI techniques that meet all but one of the 5 adaptive DRE properties and are therefore good starting points for further investigation. In general, additional research of AI techniques is needed to better understand the challenges and trade-offs and to address these deficiencies to fully support adaptive DRE systems.

VII. Concluding Remarks

DRE systems are being used for control in many important domains (*e.g.*, power grid, shipboard computing environments, manufacturing). Historically, these systems have been analyzed *a priori* to provide for provisioning resources and determining behavior. However, these systems can benefit from adaptation to leverage resources more appropriately and to respond to environments or situations not encountered previously. AI approaches can be used to enable this type of adaptation.

Many AI approaches have been developed over the years. Some of these approaches are more amenable than others relative to the properties and requirements of DRE systems. The work outlined in this paper has furthered this research in the following ways. The properties of DRE systems relative to adaptation have been clearly enumerated and defined. Several AI approaches have been summarized and taxonomized using the enumerated properties. Gap analysis has been performed to show areas of further investigation.

The AI approaches included in this work are non-exhaustive. In particular, there are hybrid approaches that include aspects of multiple traditional AI approaches (*e.g.*, neural networks and expert systems [20]). However, the taxonomy developed can be used to evaluate new or hybrid approaches for consideration with adaptive DRE systems.

References

[1] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, October 2001.

[2] B.G. Buchanan, G.L. Sutherland, and E.A. Feigenbaum. Heuristic dendral: A program for generating explanatory hypotheses in organic chemistry. In B. Meltzer, D. Michie, and R. Swann, editors, *Machine Intelligence 4*, pages 209–254. Edinburgh University Press, Edinburgh, 1969.

[3] Ming Chen. *Adaptive Performance and Power Management in Distributed Computing Systems*. PhD thesis, University of Tennessee, Knoxville, August 2010.

[4] Adnan Darwiche. Bayesian networks. *Commun. ACM*, 53(12):80–90, December 2010.

[5] Thomas G. Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Mach. Learn.*, 40(2):139–157, August 2000.

[6] E.A. Feigenbaum, B.G. Buchanan, and J. Lederberg. On generality and problem solving: A case study using the dendral program. In B. Meltzer and D. Michie, editors, *Machine Intelligence 6*, pages 160–190. Edinburgh University Press, Edinburgh, 1971.

[7] Nir Friedman, Dan Geiger, and Moises Goldszmidt. Bayesian network classifiers. *Machine Learning*, 29(2-3):131–163, 1997.

[8] Tin Kam Ho. Random decision forests. In *Document Analysis and Recognition, 1995., Proceedings of the Third International Conference on*, volume 1, pages 278–282. IEEE, 1995.

[9] Tin Kam Ho. The random subspace method for constructing decision forests. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(8):832–844, Aug 1998.

[10] Joe Hoffert, Daniel Mack, and Douglas C. Schmidt. Integrating machine learning techniques to adapt protocols for qos-enabled distributed real-time and embedded publish/subscribe middleware. *Network Protocols and Algorithms*, 2(3):37–69, 2010.

[11] Joe Hoffert, Douglas Schmidt, and Aniruddha Gokhale. Evaluating timeliness and accuracy trade-offs of supervised machine learning for adapting enterprise dre systems in dynamic environments. *International Journal of Computational Intelligence Systems*, 4(5):806–816, 2011.

[12] Jens Kober, J. Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *International Journal of Robotics Research*, 32(11):1238–1274, September 2013.

[13] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, July 1982.

[14] Yi Lin and Yongho Jeon. Random forests and adaptive nearest neighbors. *Journal of the American Statistical Association*, 101(474):578–590, 2006.

[15] JR Minkel. The 2003 northeast blackout–five years later. *Scientific American*, August 2008.

[16] Gloria Philips-Wren. Ai tools in decision making support systems: A review. *International Journal on Artificial Intelligence Tools*, 21(2):124005–1–124005–13, 2012.

[17] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[18] Sarvapali D. Ramchurn, Perukrishnen Vytelingum, Alex Rogers, and Nicholas R. Jennings. Putting the 'smarts' into the smart grid: A grand challenge for artificial intelligence. *Commun. ACM*, 55(4):86–97, April 2012.

[19] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Upper Saddle River, NJ, USA, 3rd edition, 2010.

[20] S. Sahin, M. R. Tolun, and R. Hassanpour. Review: Hybrid expert systems: A survey of current approaches and applications. *Expert Syst. Appl.*, 39(4):4609–4617, March 2012.

[21] Robert W. Sebesta. *Concepts of Programming Languages*. Pearson Higher Education, Inc., Hoboken, NJ, USA, 11 edition, 2016.

[22] G.K. Venayagamoorthy. Potentials and promises of computational intelligence for smart grids. In *Power Energy Society General Meeting, 2009. PES '09. IEEE*, pages 1–6, July 2009.