A Multiagent and Workflow System for Structural Health Monitoring Using the Contract Net Protocol and Alternatives

Gina Bullock — Dept. of Computational Sci. and Eng.
William Nick — Dept. of Computer Sci.
Kassahun Asamene — Dept. of Mechanical Engineering
Albert Esterline — Dept. of Computer Sci.
North Carolina A&T State University
Greensboro, NC, USA 27411
{gbulloc, wmnick, kasame, esterlin}@ncat.edu

Abstract

This paper reports on work on a structural health monitoring system done for NASA at North Carolina A&T State University. The system combines a multiagent system as the "brains" and a workflow system as the "brawn." Agents here typically serve as proxies for techniques with intensive communication and computation requirements. Agents negotiate to determine a constellation of techniques for solving the task at hand and communicate it to a workflow engine, which actually carries out the tasks. A hierarchically structured collection of monitor agents determines efficient ways to use the computation resources of the workflow engine. A monitor agent at the lowest level in the hierarchy is associated with a wireless mote that transmits data from an acoustic sensor (or perhaps an optic fiber or strain gauges). It negotiates with specialized agents to find techniques for extracting feature vectors from signal samples and classifying the associated events. The classifiers (in the workflow engine) are trained by several machine-learning techniques. The regions of the structure for which the various monitor agents are responsible form an inclusion hierarchy, reflected in the hierarchy among the agents. This hierarchy is set up among the agents using the contract net protocol. As alternatives, we have considered the iterated contract net protocol and coalition game theory. The current prototype architecture is restricted to a single member, but the general principles generalize to structures as complex as aircraft (the ultimate target of this project).

Introduction

The research reported here addresses monitoring the health of a mechanical structure with a multiagent system. The target structures have been aircraft, but other important targets include bridges and buildings. Structural health monitoring (SHM) (Farrar and Worden 2007) is the process of damage detection and characterization for engineering structures. Damage is defined as change to the material and/or geometric properties of a structural system. SHM involves observing the mechanical structure over time using periodically spaced measurements, extracting the damage-sensitive features and statistically analyzing these features to determine the current state of system health. SHM is used to screen the condition and provide real-time information regarding the integrity of the structure. For life critical situations, we must create a system to be the eyes, ears, and brain of the aircraft. Figure 1 shows an example of the whole system. Data is pulled from strategic areas of the plane, and information is made available to onsite and offsite technicians.

The fundamental techniques we have used so far interpret acoustic signals for clues regarding the growth of cracks, but the information on cracks in specific members must be interpreted within the context of the entire structure. Our multiagent system is a hierarchical structure that has a collection of agents that monitor the health of an aircraft structure. The monitor agents in the lowest part of the hierarchy are associated with sensors on given regions. They negotiate with agents that advocate for specialize techniques to find one suitable for classifying the events associated with the signals. The hierarchy reconfigures itself automatically, and problem solving follows the structure of the hierarchy on an as-needed basis.

In monitoring the structure of an airplane, the data must be processed not only in an intelligent and flexible way but also in near real-time. Our solution involves a multiagent system that sends a message to a workflow engine with the specification of the tasks. We set up our workflow in a sequence (pipeline) pattern where each task (stage) is its own individual process. In our setup, agents are advocates for certain computation-intensive techniques. These agents have enough intelligence so that they can work out which techniques are appropriate for the current situation and how these techniques must connect to each other in a workflow. An agent submits a multitask request to the workflow system and attaches input and output sources. So

agents are the "brains" and the workflow is the "brawn" (Foster, Jennings, and Kesselman 2004).

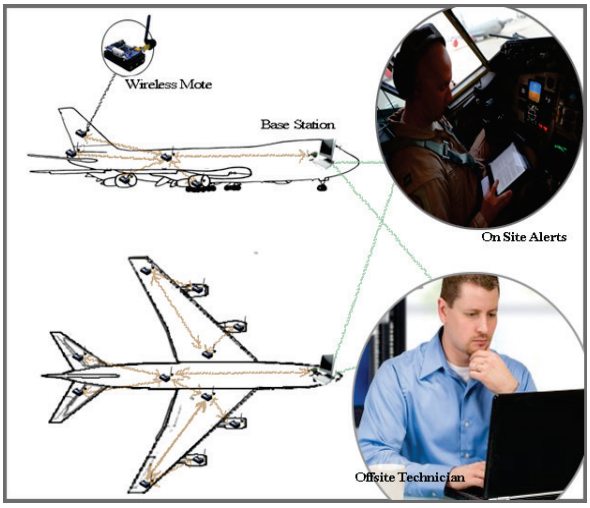

Figure1: Plane Wireless Sensor Network for SHM

The next section starts with a general introduction to multiagent systems and then discusses the contract net protocol (CNP), a problem-solving protocol for allocating system tasks to agents, which has been our main technique. The following section discusses an enhancement of the CNP called the iterated contract net protocol, and the next section introduces coalition game theory, which provides an alternative approach to forming team of agents to handle system tasks. We are starting to use the iterated CNP and coalition game theory in several places where up to now we have used the CNP. Next is a section that presents what we have in mind for the overall architecture of the monitor hierarchy, the current version, which addresses just a single member, and an abstracted view of a monitor hierarchy. The next section describes our workflow engine, and the section after that describes the use in implementing our prototype. The penultimate section explains how the CNP, the iterated CNP, and coalition game theory are applied in our SHM system, and the last section concludes.

Multiagent Systems and the Contract Net Protocol (CNP)

According to (Wooldridge 2007), multiagent systems address five main trends that have driven the advances in computing: ubiquity, interconnection, intelligence, delegation, and human-orientation. An agent is autonomous because it has control over its actions and internal state, social because it works with others in order to achieve goals, reactive because it responds to changes that occur in its environment, and proactive because it has goal-oriented behavior.

The Contract Net Protocol (CNP) is a task-sharing protocol for a collection of software agents that form the 'contract net' (Smith 1980) (Smith and Davis 1981). It was developed to specify problem-solving communication and control for nodes in a high-level distributed problem solving protocol by a negotiation process. Each agent in the network can, at different times or for different tasks, be a manager or a contractor. The CNP controls the flow of contracting and subcontracting throughout the network of agents. The CNP is designed to allow agents to break down tasks (or problems) that they cannot efficiently handle on their own into more manageable, smaller subtasks (or subproblems) using delegated agents to complete the task. Agents can bid on tasks based on their capabilities.

As portrayed in Fig. 2, in the CNP, the manager initiates the protocol by announcing the subtask, and the participants (or prospective contractors) are those agents willing to perform the subtask. The manager decomposes the problem into subproblems and *announces* each subproblem to all agents. This is a *call for proposal*s (*cfp*). All available contractors may reply with a *bid* that includes information about the resources and time needed to solve the subproblem. The manager reviews the bids and *awards* the contract to the best (from its point of view) bid. The contractor reports the solution back to the manger, which compiles a solution to the overall problem from the solutions to the subproblems.

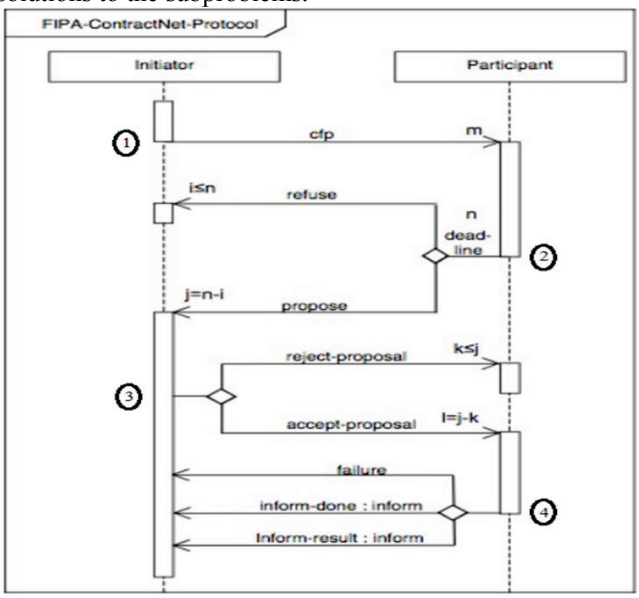

Figure 2: *Contract Net Protocol (FIPA 2002a)*

Any individual agent can operate in both roles at the same time for different tasks. If the task decomposition leads to multiple levels of subtasks, the CNP determines a hierarchy of the agents defined by the manager-contractor relation. Not only can the CNP be used to set up a hierarchy of agents to carry out a particular task over long

a period of time (such as monitoring an area of a structure), but it can also be used to solve problems on the fly.

Iterated Contract Net Protocol

The iterated contract net protocol is an extension of the basic FIPA CNP and is another FIPA standard for negotiation between agents (FIPA 2002b). The major difference from CNP is that it allows multi-rounding iterative bidding and allows multiple contractors to be chosen for a subtask (and thus team formation). From the participants who send proposals, the manager may accept the intended number and thus complete the protocol. But it may decide to issue a revised call-for-proposals to a subset of those that bid. Note that the iterated CNP requires more time, communication, and computation than the simple CNP.

In the iterated CNP (see Fig. 3), the manager issues the initial call for proposal. Participants answer with their bids.

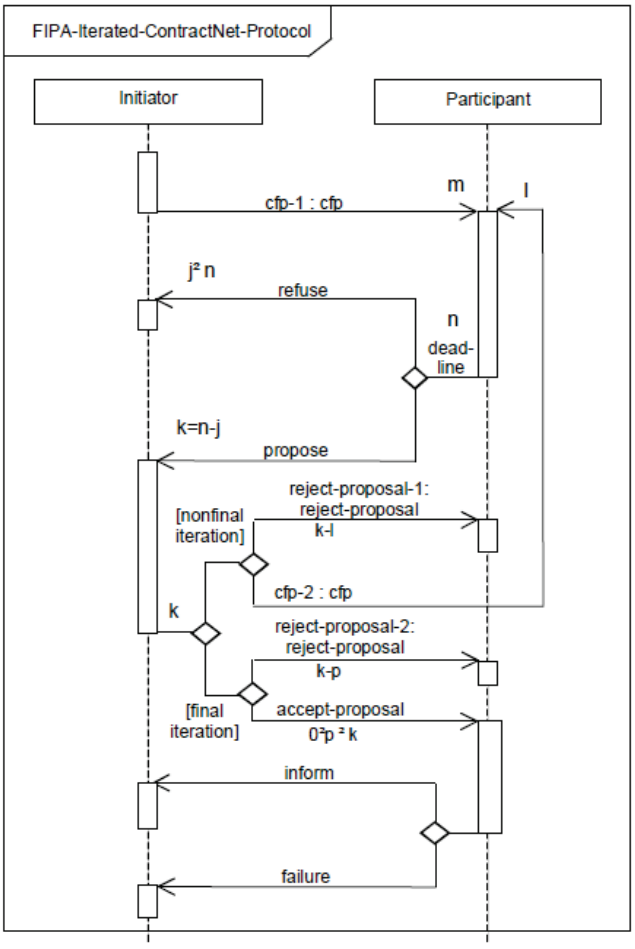

Figure 3: *Iterated Contract Net Protocol (FIPA 2002b)*

The manager has a choice to accept one or more bids and reject the others. It iterates by sending a revised CFP to get better bids. The protocol terminates if the manager declines all proposals and does not issue a new CFP, the contractors all decline to bid, or one or more bids are accepted.

Coalition Game Theory: An Alternative to the CNP

There is a greedy aspect to the CNP, which may result in clearly sub-optimal arrangements. A more radical alternative we consider is coalition (or cooperative) game theory (Osborne and Rubinstein 1994). In this, unlike the better known (competitive) game theory, binding agreements are made. Utility in the first instance accrues to groups (coalitions), not individuals. Coalition game theory addresses such questions as which are (stable) coalitions that might be formed by rational agents, and how the pay-off received by a coalition might be reasonably divided among its members.

A coalition is a set of agents, and we consider what coalitions could be formed. The key is to find how well each group can do for itself. We also consider how the coalition allocates the team's payoff to the member agents. We are not concerned with how the agents make individual choices within a coalition and how they coordinate. Instead, we take this as given and think about how all the coalitions are willing to achieve their goals.

We assume transferable utility: a coalition can redistribute the value it tries to achieve arbitrarily among members. For example: if a coalition is paid its value in money, it will be able to divide the money and to make side payments among the members in anyway.

A coalition game is a pair, (N, v), where **N** is a finite set of players and v is the *characteristic function* for the game. For every coalition $S \subseteq N$ that can be formed, $v(S)$ is the payoff that S can achieve (and divide amongst its members). An outcome of a transferable-utility game $G = (N, v)$ is a pair $(CS, \underline{x})$, where $CS = (C_1, ..., C_k)$ is a coalition structure, (i.e., a partition of N into coalitions), and $\underline{x} = (x_1, ..., x_n)$ is a payoff vector, which distributes the value to each coalition in CS. The core of a game is the set of all stable outcomes, i.e., outcomes in which no player is motivated to defect from its coalition. The core is a very attractive solution concept, but some games have empty cores.

A fair payment scheme would reward each agent according to his contribution, and this is captured by the notion of the *Shapley value* φ_i of player i. φ_i is i's average marginal contribution to the coalition of its predecessors, over all permutations of the order in which players enter coalitions. If we choose a permutation of players uniformly at random, among all possible permutations of N, then φ_i is the expected marginal contribution of player i to the coalition of his predecessors. The vector made up of φ_i for

all $i \in N$ is a payoff distribution and, in fact, this scheme is the only payoff distribution scheme with several important desirable properties.

When coalition game theory is used in artificial intelligence, probably the most pressing issue is finding an optimal coalition structure, that is, a structure with the greatest overall payoff. Approaches include dynamic programming that exploits optimal sub-structures (Rahwan and Jennings 2008) and representations of the search space that guarantee that, after a certain amount of search, the solution is within a certain bound from the optimal (Sandholm et al. 1999). Dynamic coalition formation has been studied as a Markov chain (Dieckmann and Schwalbe, 1998).

The CNP, a classical multiagent negotiation model, was proposed in 1980 and focused on how to allocate the system tasks in order to solve conflicts over resources and knowledge. A major problem with the CNP is that system load rises rapidly as the scale of the issues increases. Also, negotiation quality is not maintained across task environments. The CNP can be combined with a coalition mechanism to contain negotiation costs and to stabilize the quality of the structure of the multiagent system.

Monitor Hierarchy

Fig. 4 depicts the ultimate goal of a multiagent system for the SHM for an aircraft. Agents are given dedicated roles within the system to carry out needed tasks at various levels.

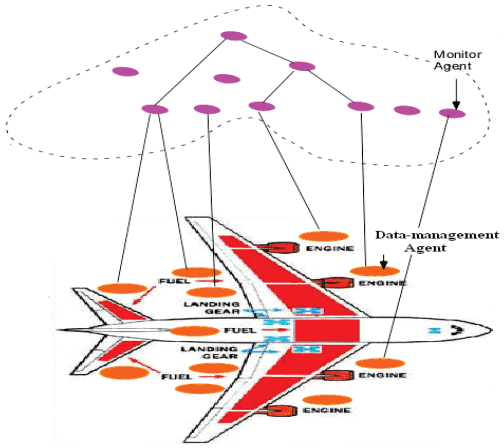

Figure 4: Overall Architecture

The current, less ambitious architecture involves a single member, as in the schematic diagram in Fig. 5. The two kinds of sensors are piezoelectric (PZT) sensors (which record acoustic waves and so detect cracks forming at a distance) and optic fiber (which captures the strain distribution across its length). We may include strain gauges. Sensors are distributed over the plate, and there are more sensors than there are observed data streams. Agents determine what data streams are used in response to inferred conditions. The different kinds of sensors are combined in intelligent ways, and areas are combined hierarchically.

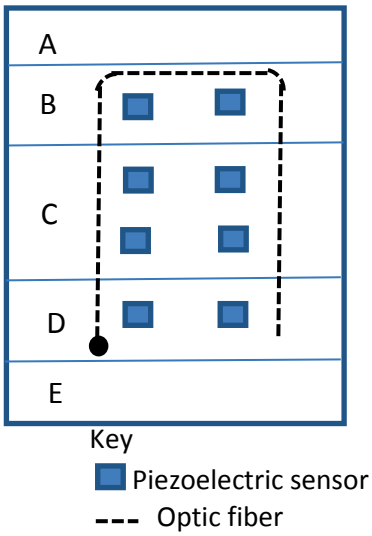

Figure 5: Current Architecture: Single Member with Sensors

Each active data stream is associated with a data agent as a sort of gatekeeper. The data agent is involved in an instance of the CNP in which it is paired with a monitor agent that willbe responsible for interpreting the data stream. This monitor agent uses the CNP to find an agent advocating for a classification technique appropriate for the current context. The classification agents involved are proxies that advocate for various techniques implemented in the workflow engine. With the classification technique determined, the monitor agent then uses the CNP to find one or more feature extraction techniques to extract vectors of features from the data stream to provide for the classifier. Again, the feature-extraction agents are proxies for techniques implemented in the workflow.

The monitoring hierarchy controls the flow of information between the lower levels and the upper levels and thus allows the system to manage large amounts of streaming time-sensitive data in an efficient manner. Having data in a timely manner allows the system to accurately refine the data and classify the events. Once the data has been classified, the system can facilitate situation awareness by comparing one or more classifications.

A complex structure (e.g., an aircraft body, even a large plate) is hierarchically organized. Monitor agents monitor components in the hierarchical structure of the aircraft. The CNP is used initially to allocate a hierarchy of areas of responsibility. This reflects the hierarchy of the monitored structure. The hierarchy of monitor agents changes dynamically to focus on suspect areas. A simple strategy for this is to drop a monitor agent from where they are

underused and to introduce a monitor agent where more attention is needed. In either case, the CNP must be run again, and a major issue is to limit the extent of the hierarchy that may thereby get reconfigured. Monitor agents also use the CNP to address problems on the fly, with no impact on the hierarchical structure. Analysis tools are available in the workflow engine for diagnosis and prognosis of members and multi-member structures.

In the general schema sketched in Fig. 6, top-level monitor agents are in direct contact with the users. Those at the bottom have the most direct access to the data via data agents. Monitor agents higher up provide larger-scope and less detailed information. Some issues can be addressed locally. Others require data/information from multiple areas. Alerts are passed up in the hierarchy. Data is requested for the particular sensor node by sending a request down to a particular data agent.

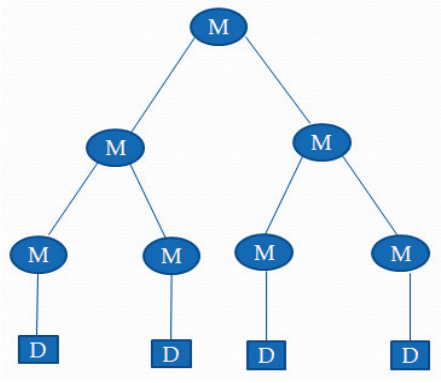

Figure 6: Monitor agents (M) and data agents (D) in a hierarchy

Workflow Engine Architecture

Our current prototype workflow engine (see Fig. 7) handles the processing of a datastream in terms of the selected feature extraction and classification techniques. It is in the tradition of scientific workflows (Singh and Vouk 1996) although it uses web standards cautiously so that large volumes of time-critical data may be handled, and the computational resources are considered part of the workflow engine. The techniques used are determined by two instances of the CNP. After the two instances of the CNP have finished, a monitor agent sends the name of the feature extraction and classification techniques to the data agent. The data agent sends the names of these techniques to the workflow engine through web services. Then the workflow engine puts the ID of the job, the IP address or host name of the machine running the workflow engine, and the port number of the machine into a NoSQL database. Then the workflow engine sends the ID of the job to the data agent. The data agent wakes up the input and output daemon processes, sending them the ID of the job. The input and output daemon processes, using the ID of the job as a key, query the database for the address of the machine running the workflow engine and its port number. The input daemon reads the data from the mote (see below) and writes it to a UDP socket. We use a UDP socket for input because it is faster than TCP since it tolerates anomalies in the input. A *mote* is a wireless sensor network node capable of performing some processing, gathering sensor data, and communicating with other nodes in the network. The last process in the workflow writes information to a TCP socket, and the output daemon reads information from this TCP socket and writes it to a database, where it can be accessed by the multiagent system or any other system or user. TCP sockets are used for output so that information gets to the output daemon reliably. With our workflow system, multiple workflows can run at the same time, and a given workflow can run multiple tasks in parallel.

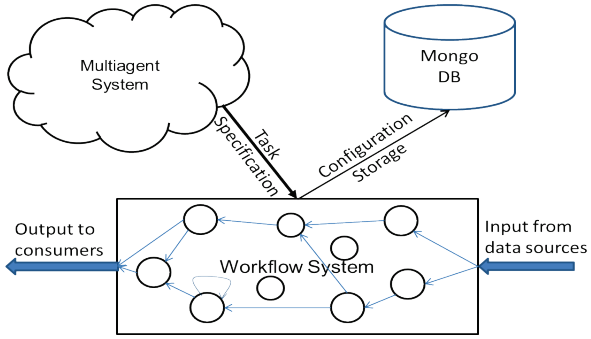

Figure 7: The workflow system in context

Fig. 8 focuses on the data flow in a given workflow that has been set up, viewed as a single pipeline. In fact, the feature extractor, or (better) feature master, controls one or more slaves, depending on the choice of feature extraction techniques—see Fig. 9. The output of each slave (sub-vectors of the overall feature vectors) goes into a buffer, where the feature master assembles the full feature vectors by matching timestamps. The full vectors are written on the pipe to the classifier process.

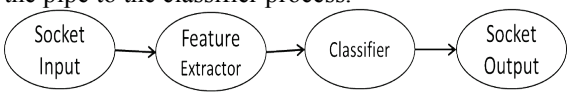

Figure 8: Workflow structure for classifying acoustic events, setup using the sequence pattern.

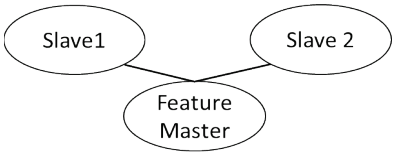

Figure 9: Example of the feature master and its slaves

The classification techniques we have used are trained with a variety of machine-learning techniques (Bishop 2006) (Duda, Hart, and Stork 2001) with and without dimensionality reduction. The dimensionality reduction

techniques investigated include principal component analysis (PCA), recursive feature elimination (RFE) and Genetic and Evolutionary Feature Weighting & Selection (GEFeWS) (Alford et al. 2011). The unsupervised learning techniques used are k-means and self-organizing maps (SOM), and the supervised learning techniques are support vector machines (SVM), naive Bayes classifiers, and feed-forward neural networks (FNN).

Development of the workflow engine has concentrated so far on interpreting single data streams, corresponding to monitor agents at the lowest level of the hierarchy. Future work will address the greater scope associated with monitor agents higher in the hierarchy. Computational techniques at these higher levels address the monitored structure itself and include such things a finite-element analysis tools.

Implementation

We used the Java Agent Development Environment (JADE) framework (Bellifemine, Caire, and Greenwood, 2007) to implement our agent system. JADE was preferred over other agent frameworks because it is the most widely used and mature agent framework and uses FIPA standards. (FIPA (IEEE FIPA Standards Committee 2014) is an IEEE Computer Society standards organization that promotes agent-based technology and the interoperability of its standards with other technologies.) It also provides implementations of the CNP and the iterated CNP. JADE allows communication between agents either on the same or different platforms. JADE agents can expose their services as web services using the WSIG (Web Services Integration Gateway) add-on (JADE Board. 2011).

For the database used to store data and workflow configurations, we used the MongoDB engine, which is a document-oriented NoSQL database management system that can run on many platforms (MongoDB, Inc. 2014). A MongoDB collection is a grouping of documents. MongoDB's collections do not enforce document structure before inserting data, which makes it excellent for storing large amounts of data. It stores data in a JSON-like format called BSON (Binary JavaScript Object Notation), which is primarily used as a data storage and network transfer format. In MongoDB, BSON objects are also called documents.

The input and output daemons were written using the Apache Daemon API for Java (Apache Commons 2013), which is a part of the Apache Common suite. The ring buffer, which is used between the feature master and its slaves, was written in C using the Python API for C (Python Community 2015). The Python API for C allows compiled C code as a Python module. We are using a part of a Python machine learning package called scikit-learn (Pedregosa et al. 2011) to train the classification techniques. The scikit-learn package is built on the NumPy (Oliphant 2007), SciPy and Matplotlib packages. Our feature extractors are implemented using NumPy and SciPy. NumPy is a Python extension for numerical computing. SciPy is an open source Python library that is built on top of NumPy for scientific computing.

Application of the CNP, Iterated CNP, and Coalition Game Theory

This section sketches how the CNP, the iterated CNP, and coalition game theory have been used in our prototype. We consider the simple case of the plate with sensors attached depicted in Fig. 5.

Application of the CNP

The monitor agents have access to the schema of the plate: locations of devices, functional blocks, spatial layout, and the likelihood of crack initialization as a function of the location on the plate. There are three subtasks here: monitor the plate where it is supported at A and again at B, and monitor it in the middle, at C. The middle is the most likely region for crack initialization, but the regions where the plate is supported are also vulnerable. The sensors in B provide the best data on region A although the top two sensors in C can provide some information. Similarly, the sensors in D provide the best data on E although the bottom two sensors in D provide some.

The topmost monitor agent first announces the task of monitoring region C. The agents whose bids do well here are proxies for computational tasks that combine several classification streams from PZT data and strain data from the optic fiber. The top monitor agent then announces the subtask of monitoring one of the support areas, say, A. The monitor agents that will do well here will be ones good at combining a small number of classification streams from PZT sensors but are able to add a few more streams if required. (The optic fiber is not useful for monitoring A or E since it does not record waveforms.)

After a certain monitor agent is awarded the task of monitoring region C, it determines the information it needs and issues announcements for classification streams. Various monitor agents appropriate for classifying events from streams of PZT data associate with the various data agents paired with the sensors. These monitor agents will participate in the CNP for classification streams. One agent-stream pair might be enough initially, or the monitor agent for C might decide two are needed, and initiate another CNP instance. The monitor agents for regions A and E similarly perform instances of the CNP to award monitoring tasks to one or two monitor-senor pairs each.

If one of the classification streams from region C suggests crack formation, the monitor agent for C might

initiate another instance of the CNP to obtain another classification stream. Likewise, if one of the classification streams in B or D suggests crack formation in A or E, the monitor agent for A or E might try to get another data stream. If the situation is critical, one of these monitor agents may allocate a third agent-sensor pair, taking a neighboring one from region C. If things are quiet, the monitor agent for a region may release an agent from an agent-sensor pair to conserve resources. The sensor thus freed may be needed by a monitor agent for another region. Where there is contention, the parent (or lowest common ancestor) of the two contending monitor agents decides.

Two monitor agents higher in the hierarchy may tap information from the same agent lower in the hierarchy; in this sense the so-called hierarchy is not tree-shaped. But certain control relations are maintained (e.g., requesting specific data), and each agent that does monitoring has a unique parent that is the ultimate arbiter of what it may do.

If all contractors persist throughout (i.e., there is no notion of completing the task) and there are more subtasks than agents in the system, then some subtasks will go unassigned. For performance, we do not allow an unbounded number of agents. We can try various policies for creating additional agents as needed, or we can have a policy for which tasks may be left unassigned.

Note that, even with non-overlapping regions, there may be several concurrent instances of the CNP. This happens when one agent is the manager for several subtasks. It completes the CNP for the first subtask and moves on to the others. Meanwhile, the contractor awarded the first subtask assumes its role as manager for the subtasks in it. We would like some indication of how severely this impacts performance and possibly restrict the number of concurrent instances of the CNP.

Application of the Iterated CNP

Consider again the case where the monitor agent for region C initiates instances of the CNP to allocate agent-sensor pairs. This is an opportunity for the iterated CNP. The monitor agent could accept several bids in light of which it could refine its cfp and eventually accept the one or several agent-sensor pairs required. Note that this may avoid sub-optimal allocations due to the greedy nature of the CNP.

In our prototypes, each agent has a hash with the IDs of the other agents with which it might cooperate as keys and real numbers as values. Such a value represents how well the agent collaborates with the agent identified by the key. In the first iteration, the cfp from the manager includes just the functional region, and all agents that cover part of it submit proposals (identifications and what they contribute). In the second iteration, the manager sends a cfp to only the agents who submitted proposals on the first iteration. The cfp includes the IDs of all the agents who submitted proposals. A proposal now includes the agent's functional region and the part of its hash with the other submitting agents. The manager now selects a subset P of the proposers that covers the region and maximizes an expression that is a sum $A + B$, where $A = M/|P|$ (M a constant—note that small $|P|$ results in big A) and $B = \sum_{a \in P} \sum_{b \in P, b \neq a} H_a(b)$. (Here H_a is the hash for agent a.)

Application of Coalition Game Theory

Using coalition game theory, we work from the bottom up. This is somewhat like the approaches (such as (Rahwan and Jennings 2008)) that have used dynamic programming to construct optimal coalition structure (i.e., the structure with the greatest overall value) from optimal solutions to subproblems. In our case, however, possible lowest-level coalitions are severely constrained by the need to form teams of agents with complementary competences. The payoff for a coalition is some measure of its contribution to the overall assurance of the integrity of the monitored structure minus a measure of the resources used. For the plate in Fig. 5, we would expect that, for a structure with the greatest overall value, the sensor-agent pairs in B would form one coalition, those in D would form another, and those in C would form a third. Since the coalition structure will be hierarchical, each of these coalitions will include an agent that works as their representative. This is like the monitor agent in the CNP that served as the manager for the contractors that are now seen as members of the same coalition. Using the iterated CNP, we assume that the agents have some measure of how well they collaborate with the other agents they might be expected to work with. With the coalition-game-theory approach, the agents need considerably more information (such as the current context and exactly what competences other agents provide) since there is no direction from a manager agent.

Conclusion

We have described our prototype system where a hierarchically structured collection of monitor agents monitors the health of a structure by interpreting acoustic signals. A monitor agent at the lowest level negotiates with specialized agents to find techniques for extracting feature vectors from signal samples and classifying events associated with signals. A monitor agent higher in the hierarchy interprets the state of the structure covered by the sensors associated with its descendant agents. The hierarchy reconfigures dynamically, and problem solving, guided by the contract net protocol, follows the structure of the hierarchy in opportunistic ways. Agents here typically serve as proxies for techniques with intensive communication and computation requirements.

Our multiagent system sends a message to a workflow engine with the specification of the tasks. A workflow has a pipeline pattern where each stage is its own process.

Agents are advocates for certain computation-intensive techniques and work out which techniques are appropriate for the current situation and how these techniques must connect to each other in a workflow. An agent submits a multitask request to the workflow system and attaches input and output sources. So agents are the "brains" and the workflow is the "brawn." The classification techniques are trained with a variety of machine-learning techniques. We have concentrated on interpreting single data streams, corresponding to monitor agents at the lowest level of the hierarchy. Future work will address the greater scope associated with monitor agents higher in the hierarchy, where the corresponding computational techniques address the monitored structure itself.

The contract net protocol (CNP) takes a greedy approach to allocating subtasks: subtasks are allocated to the best potential contractor without considering the subtasks subsequently allocated. Better results can be obtained with the iterated CNP, which can allocate sets of subtasks to sets of agents and do more than one round of announce-and-bid before making an allocation. Coalition game theory is potentially better still as it provides rigorous ways to allocate constellations of task to constellations of agents in ways in some sense optimal. We are pursuing both the iterated contract net protocol and coalition game theory in the context of the system described here.

Acknowledgements

The authors would like to thank the Army Research Office (proposal number 60562-RT-REP) and NASA (Grant # NNX09AV08A) for financial support. Thanks are also due to members of ISM lab (and Dr. Mannur Sundaresan of the Mechanical Engineering Dept.) at North Carolina A&T State University for their assistance.

References

Alford, A., Popplewell, K., Dozier, G., Bryant, K., Kelly, J., Adams, J., Abegaz, T., and Shelton, J. 2011. GEFeWS: A Hybrid Genetic-Based Feature Weighting and Selection Algorithm For Multi-Biometric Recognition. In *Proc. 22nd Midwest Artificial Intelligence and Cognitive Science Conference* (*MAICS 2011*), 86-90. Red Hook, NY: Curran Associates, Inc.

Apache Commons 2013. Commons Daemon, Apache Software Foundation. Retrieved 1-27-2015, http://commons.apache.org/proper/commons-daemon/.

Bellifemine, F.L., Caire, G., and Greenwood, D. 2007. *Developing Multi-Agent Systems with JADE*. Hoboken, NJ: Wiley.

Bishop, C. 2006. *Pattern Recognition and Machine Learning*. New York, NY: Springer.

Dieckmann, T.; Schwalbe, U. 1998. Dynamic Coalition Formation and the Core. Maynooth, County Kildare: National Univ. of Ireland, Maynooth.

Duda, R., Hart, P., and Stork, D. 2001. *Pattern Classification*, 2nd ed. Hoboken, NJ: Wiley.

Farrar, C.R. and Worden, K. 2007. An Introduction to Structural Health Monitoring. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 365(1851):303–315.

FIPA 2002a, FIPA Contract Net Interaction Protocol Specification. Retrieved 2015-02-20 from http://www.fipa.org/specs/fipa00029/SC00029H.html

FIPA 2002b, FIPA Iterated Contract Net Interaction Protocol Specification. Retrieved 2015-02-20 from http://www.fipa.org/specs/fipa00030/SC00030H.pdf

Foster, I., Jennings, N. R., and Kesselman, C. 2004. Brain Meets Brawn: Why Grid and Agents Need Each Other. In *Proc. of the Third International Joint Conference on Autonomous Agents and Multiagent Systems* (*AAMAS'04*), 1:8-15. New York, NY: ACM.

IEEE FIPA Standards Committee 2014. Welcome to FIPA! IEEE Computer Society. Retrieved 1-27-2015 from http://www.fipa.org/.

JADE Board 2011. JADE Web Services Integration Gateway (WSIG) Guide. Retrieved 2-18-2013 from http://jade.tilab.com/doc/tutorials/WSIG_Guide.pdf.

MongoDB, Inc. 2014. MongoDB. Retrieved 1-27-2015 from http://www.mongodb.org/.

Oliphant, T. E. 2007. Python for Scientific Computing. *Computing in Science & Engineering* 9(3)10-20.

Osborne, M. J. and Rubinstein, A. 1994. *A Course in Game Theory*. Cambridge, MA: MIT Press.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg,V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12:2825-2830.

Python Community 2015. *Python/C API Reference Manual* (ver. 2.7). Python Software Foundation. Retrieved 1-27-2015 from https://docs.python.org/2.7/c-api/.

Rahwan, T. and Jennings, N. R. 2008. An Improved Dynamic Programming Algorithm for Coalition Structure Generation. In *Proc. of the 3rd Int. Joint Conf. on Autonomous Agents & Multiagent Systems* (*AAMAS'08*), 1417-1420. NYC, NY: ACM.

Sandholm, T., Larson, K., Andersson, M., Shehory, O., Tohme, F. 1999. Coalition Structure Generation with Worst Case Guarantees. *Artificial Intelligence* 111(1–2):209–238.

Singh, M. P., and Vouk, M. A. 1996. Scientific Workflows: Scientific Computing Meets Transactional Workflows. *Proc. of the NSF Workshop on Workflow and Process Automation in Information Systems: State-of-the-Art and Future Directions*. Retrieved 1-27-2015 from http://www.csc.ncsu.edu/faculty/mpsingh/papers/databases/workflows/sciworkflows.html.

Smith, R. G. 1980 The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver. IEEE Transactions on Computers, C-29,12:1104-113.

Smith, R. G. and Davis, R. 1981. Frameworks for Cooperation in Distributed Problem Solving. *IEEE Transactions on Systems, Man and Cybernetics*, 11,1:61 – 70.

Wooldridge, M. 2009. *An Introduction to MultiAgent Systems*. Hoboken, NJ: Wiley.