A Framework for Learning Coordinated Behavior

Albert Esterline
Comp. Sci., N. Carolina A&T SU
Greensboro, NC 27411, U.S.A.
1-336-334-7245
esterlin@ncat.edu

Chafic BouSaba
Abdollah Homaifar
Elect. & Comp. Eng., NC A&T SU
Greensboro, NC 27411, U.S.A.
1-336-334-7760
{cbousaba,homaifar}@ncat.edu

Dan Rodgers
R. & D. Dept., General Dynamics
Austin, TX, U.S.A.
1-512-264-8782
drodgers@gdrs.com

ABSTRACT
We sketch a framework for learning structured coordinated behavior, specifically the tactical behavior of Experimental Unmanned Vehicles (XUVs). We conceptualize an XUV unit as a multiagent system (MAS) on which we impose a command structure to yield a holarchy, a hierarchy of holons, where a holon is both a whole and a part. The formalism used is a conservative extension of Statecharts, called a Parts/whole Statechart, which introduces a coordinating whole as a concurrent component on a par with the coordinated parts; wholes are related to common knowledge. We use X-classifier systems (XCSs). Exploiting Statechart semantics, we translate Statechart transitions into classifiers and define data structures that interact with an XCS.

Categories and Subject Descriptors
I.2.6 [**Artificial Intelligence**]: Learning—*knowledge acquisition*. D.2.2 [**Software Engineering**]: Design Tools and Techniques—*State diagrams*.

General Terms
Design, Theory.

Keywords
Coordinated behavior, Parts/whole Statecharts, XCS, Semantics.

1. INTRODUCTION
This paper sketches a framework for learning highly structured coordinated behavior in the tactical behavior area of Experimental Unmanned Vehicles (XUVs). We conceptualize a unit of cooperating XUVs as a multiagent system (MAS) [7]. The hierarchical structure of military command and control is imposed on the underlying MAS to yield a holarchy [8], a hierarchy of holons, where a holon is both a whole and a part. Parts/whole Statecharts, a conservative extension of Statecharts, is used as the formalism [10]. Since the implicit coordination of normal Statecharts eliminates encapsulation, a Parts/whole Statechart introduces a coordinating whole as a concurrent component on a par with the coordinated parts. The learning technique we use is X-classifier systems (XCSs) [11], where learning consists in acquiring a population of weighted condition-action classifier rules that direct behavior. By exploiting Statechart semantics, we translate Statechart transitions into classifiers and define data structures that support a simulation that interacts with an XCS.

The remainder of this paper is organized as follows. Section 2 introduces the standard notions of MAS and holarchy; it also presents Parts/whole Statecharts. In section 3, we present the translation of Statecharts to XCS. Section 4 develops the XCS operations and is followed by GA operations. Section 6 discusses the learning of wholes. Section 7 concludes and discusses future work.

2. PARTS/WHOLE STATECHARTS, HOLARCHIES
A MAS is a group of autonomous agents cooperatively accomplishing complex tasks that any individual agent alone is incapable of [7] by a knowledge-level communication. Complex coordinated operations require social organization about which the general MAS paradigm is mute. Classically, social institutions with efficient communication and control have hierarchical structures. To analyze the coordination hierarchy of agents, we use Koestler's notion of a *holarchy* [8], which is a hierarchy of holons. "*Holon*" is a combination of the Greek word "holos," meaning whole (self-assertive), and the suffix "on," meaning part (cooperative).

A holarchy is a concurrent structure. Due to the difficulty of the analysis of concurrent systems, it is thus highly desirable to use rigorous methods in specifying, designing, and testing them. Of the various formal methods for modeling concurrent systems, automata are the most concrete and hence most accurately reflect the operations of implemented systems. The most popular automata for modeling concurrent systems are Statecharts [5], a visual formalism for the behavioral description of complex systems that extends classical state diagrams in several ways.

A Statechart lets us represent hierarchies of states and concurrently active states. A superstate that contains substates and transitions that elaborate its sequential behavior is called an XOR state since, when it is active, exactly one of its substates is active. A superstate with concurrent substates—"orthogonal components"—is called an AND state. A basic state has no substates so is neither an XOR nor an AND state. A transition has a label e/a indicating that, when the system is in the source state and event e happens, it can move to the target state on performing action a; the action part is optional. One substate of an XOR state

GECCO'05, June 25-29, 2005, Washington, DC, USA.

is identified as the default state, the substate that becomes active when a transition is made to the XOR state. A transition may have a non-basic state as either its source or target. In fact, it may have several source states or several target states. A transition label more generally may be of the form e[c]/a (a again being optional), where c is a condition that must be true for the transition to fire. Frequently, the condition indicates that an orthogonal component is in a given state. The hierarchical structure of a Statechart makes it easy to suppress detail (*zoom out*) or to focus on detail (*zoom in*). The Statechart formalism is characterized not only by hierarchy (XOR states) and orthogonality (AND states) but also by broadcast communication: an action in one orthogonal component can serve as a triggering event in other components. This is one way components of a Statechart are implicitly coordinated. Others ways are by transition conditions that test the state of another component and by using the same event as a trigger in transitions in different orthogonal components. This implicit coordination introduces a web of references and eliminates encapsulation, adversely impacting the reusability, understandability, and extendibility of the resulting abstractions. We therefore follow Pazzi [10] in using an explicit approach to modeling aggregate entities, introducing a *whole* as an orthogonal component on a par with the parts that are coordinated. A part communicates only with the whole, never with other parts. The resulting *Parts/whole Statecharts* introduce no new notation but simply put constraints on how orthogonal components coordinate and require a specific section for the whole and different sections for the parts. A parts/whole Statechart naturally supports two kinds of hierarchies, XOR hierarchies and parts/whole hierarchies, the latter modeling holarchies. In fact, the Parts/whole extension of Statecharts is the perfect modeling technique to represent the holons' most fundamental characteristics, the "self-assertiveness tendency" and the "integrative tendency".

When we model a holarchy, since the purpose of the whole is coordination, it is reasonable to consider it as corresponding to *common knowledge*, which is a necessary condition for coordination [4]. We can also characterize common knowledge in terms of shared situations [2]. Barwise [2] concludes that the fixed-point approach is the correct analysis of common knowledge, but that common knowledge generally arises via shared situations. Implementing a part endows an individual with the capacity for certain public behavior so that it may realize the role described by the part. To avoid an infinite regress, however, wholes are not implemented apart from the individuals instantiating the parts. So the whole must be instantiated in each part. The parts must be synchronized so that their instances of the whole are always the same. Each whole instance interacts with its co-instantiated part instance and with the part instances instantiated in the other parts.

3. TRANSLATING STATECHARTS TO CLASSIFIERS

To apply an XCS to a Statechart, we must translate the Statechart into classifiers with a well-defined structure. To keep things simple, we restrict ourselves to transitions that have a single source state, a single target states, and labels of the form *e*[in *s*]/*a*, where *a* is optional. We also ignore history symbols and structural features other than XOR, AND, and basic states and transitions of form just mentioned. The basic idea, then, is that a classifier represents a transition (or a class of transitions if there are *don't care*s in the condition). One step through the XCS cycle corresponds to taking one transition in the Statechart, although the actions performed in the step must be remembered for any transitions they might subsequently trigger (all transitions in the current *micro step* in the sense of [6]). The format for a classifier, then, is

<source state><event><cond state>:<action><target state>

Each classifier part is called a *role*. There are two *categories* of elements that roles denote: states and acts. The <source state>, <cond state>, and <target state> roles denote states, while the <event> and <action> roles denote acts. The condition in this classifier, then, is the source state of the transition and the event and condition from the transition's label. The consequent is the action from the transition's label and its target state.

Each of the three roles in the condition can be a *don't care*, but a role cannot be encoded with some positions occupied by *don't care*s and some not. So no partial matching on roles is allowed. Note that a don't care in the <cond state> role corresponds to the absence of a condition in the transition label.

For applying the GA part of the XCS, it is not enough to capture just the structure. This is because the event and action on a transition generally have a semantic relation to the source and the target state. The event is something that can happen when in the source state, causing a change to the target state, and the action is something that can be done in the source state, ending that activity and initiating the activity represented by the target state. In addition, there is also a semantic relation between the event and the action: the action is an appropriate response to the event (when we are in the source state). Therefore, each state and act is associated with a feature vector. Each feature has a value in {0,1} and is represented by a unique position in the feature vector. There are consistency rules for each category, which forbid certain combinations of features. There are also agreement rules, analogous to the agreement rules in natural languages (e.g., that subject and verb must agree in person and number), which restrict the features a state or act may have in a given role in terms of the features of the elements filling the other roles.

4. XCS OPERATION

To carry out the operation of the XCS, besides the classifiers, we need to store general information about each state, which we assume is available in a *state table* indexed by state names. For each state, this records its type (the ψ function of [6]), its children (the ρ function of [6]), and its feature vector. For an XOR state, the state table also records its default state (the δ function in [6]). In the next subsection, we mention several other data structures for feature vectors.

The operation of the XCS is most easily pictured when the environment is a simulation that maintains a set of data structures that relate directly to Statechart semantics. The detectors then simply read from these data structures and the effectors simply write to them, and the simulation performs whatever adjustments are needed to these data structures so that they conform to Statechart semantics. To begin with, we need to record the set of acts corresponding to actions that have been performed and are still available as triggering events; we call this the *active set*. As mentioned, an act may reside in the active set for more than one

XCS cycle. The active set also includes events that occur in the environment of the Statechart. We must also record the set of basic states currently active. This corresponds to the partial state configuration in [6] that is a component of the micro system configuration; we refer to it, however, as the *state configuration*. To find the set of all active states (not just the active basic states), we use a table, the *foundation table*, that associates with each non-basic state the set of its basic descendants.

When the XCS is implemented in a group of agents corresponding to the Statechart, each agent runs an instance of the XCS procedure. Each agent is viewed as an instance of a part in the holarchy and has its own instances of the whole of which it is an immediate part and of all the wholes above it. Each agent has states that, along with the states of the other agents, contribute to the environment of the instance of the XCS procedure run by that agent. The environment itself in this sense maintains all the information that in simulation is maintained by the active set, the state configuration, and the foundation table.

In the remainder of this subsection, we focus on the case where the system is run in simulation, and we address only aspects of the XCS that are specific to Statecharts: handling non-determinism, forming the match set, what to do once a consequent is selected, and reward evaluation. The more involved issue of the operation of the GA is deferred to the following subsection and issues relating to coordination and wholes are deferred to the section after that. To begin with, then, non-determinism (as when there are two transitions with the same source state and triggering event but no distinguishing condition) is an important aspect of Statecharts. The random selection that is built into XCSs for exploration can be used here as well.

As regards forming the match set, a condition matches if (1) the state in its <source state> role is in the state configuration or has a descendant in the state configuration (as determined by the foundation table), (2) the act in its <event> role is in the active set, and (3) the state (if any) in its <cond state> role is in the state configuration or has a descendant in it.

Next, when a consequent is selected, the act in its <action> role is added to the active state. We must also remove from the state configuration the basic states that are descendants of the state filling the <source state> role. These are found by recursively querying the state table. Finally, we must add to the state configuration the basic states descended from the state in the <target state> role that become active when the corresponding transition is taken. For this, we construct a set of states, initialized to contain only the target state, by recursively querying the state table. When we come to an AND state, we remove it from the set and add its children. When we come to an XOR state, we replace it with its default state. The procedure terminates when only basic states remain in the set.

Finally, following [6] we take a *run* to consist of a sequence of transitions that begins with the initial configuration, where the state comprising the entire Statechart (the root) is entered. Statecharts of interest to us, unlike many, model systems expected to terminate—the mission is accomplished, aborted, cancelled, or modified. We therefore restrict the notion of a run further by requiring that it end with a final configuration. Note that, because of both non-determinism and the GA, there may be several alternative runs even in the same environment. Part of the reward evaluation is assigning points for the final states, but various characteristics also contribute—e.g., how long it took, resources used, risks taken, and opportunities lost. The overall reward for a part should include something for the performance of its unit, the larger unit of which it is a unit, and so on. How a unit's performance contributes to a part's reward, however, is a complex issue since being part of a poor unit may actually enhance our judgment of a reasonable agent.

5. GA OPERATION

Since the action set contains only classifiers that advocate the same consequent, crossover has an effect only on the condition parts of classifiers, although mutation can affect both the condition and the consequent of a classifier. Since the names of states and acts are largely arbitrary, and because of the semantic relations among the elements that occur in the roles in classifiers, we apply crossover and mutation to the feature vectors for these elements. We assume that there is an isomorphism between act names and act feature vectors. Thus, given an act name in a classifier, we can get its feature vector to perform crossover and mutation, and, given a feature vector that results from crossover or mutation, we can get the associated name to insert into the new classifier. In contrast, there might be several states with the same feature vector since states can be distinguished by where they occur in the Statechart.

A new classifier with only an act feature vector changed essentially adds this transition to the Statechart as long as the feature vector is consistent. The transition, however, may disrupt coordination with a whole. A straightforward way out of this problem is to record which acts are used for coordination and to block changes to such acts in classifiers. The alternative is to invoke a repair procedure that updates the whole and the other parts that coordinate with the whole.

When we have a new classifier with a state feature vector changed, the feature vector may be associated with one or more existing states, or there may be no state with that feature vector. If there is exactly one state, then a new transition to or from that state is added. If there are several states, then we can choose a state closest (in terms of the structure of the Statechart) to the other state in the classifier. If this rule does not determine a unique state, then we can turn to the feature vectors of elements filling other roles in the classifier and apply a rule based on semantic distance. If the modified state feature vector in a new classifier is associated with no existing state, we must create a new state, which generally would be in the same orthogonal component as the state filling the other state role in the new classifier. Where to place the state could again be determined in terms of structural and semantic distance.

When a new state is added, it initially occurs in only one classifier or, equivalently, takes part in only one transition (as either the source or target). If it is a target, then there is no way out of it. If it is a source, there is no way into it. A procedure similar to covering could introduce a new, rather general classifier corresponding to multiple transitions to or from the new state.

There are many issues that could be addressed regarding modifications a GA may make to the structure of a Statechart. Some changes, such as changing an XOR to an AND state or vice versa, are clearly forbidden. Certain large-scale changes have

significant intuitive appeal. For example, moving a large coherent part of the substructure of one state to another corresponds to assigning the responsibility for part of an activity to another agent. Space restrictions prohibit exploration of such issues.

6. LEARNING WHOLES

If we consider the XCS applied to a Parts/Whole Statechart in simulation, learning coordination behaviors is apparently no more difficult than learning other behaviors except for changes caused by the GA. Such changes may modifying structure and generally require modifications to the behaviors of the parts. Accommodating these kinds of changes is similar to modifications required when goals are elaborated in some approaches to parallel planning. There has also been relevant work in game theory on communication and coordination protocols appropriate in various strategic situations [3]. In general, restructuring the coordination structure is a large problem that can be related to several relatively mature approaches.

If we consider the XCS implemented in a system modeled with a Parts/Whole Statechart, learning coordination behaviors is much more difficult since we have a copy of the whole for each group member. We can frame this problem in terms of acquiring new common knowledge, and a formal analysis reveals that this is surprisingly difficult [4]. In a controlled setting, we can generally arrange for checks, but it is obvious that a dynamic environment over which we have little control is not good for learning coordination strategies. A particular problem is how the group may agree on an innovation, and it is easy to show that agreement implies common knowledge [4]. Interest here is perhaps more with innovation, hence flexibility and autonomy, than with learning. This is accommodated by associating wholes with common knowledge rather than seeing coordination as the coherence of behaviors assigned to agents.

7. CONCLUSION

We consider a group of XUVs learning military tactical behaviors, especially the coordination of units, given a formal representation of the behavior of the group. We conceptualize the group as a multiagent system and view the control structure as a holarchy. The formalism used is an extension of the Statechart notation, called Parts/whole Statecharts [10], which introduces a coordinating whole as a concurrent component on a par with the coordinated parts. We relate wholes to the technical notion of common knowledge, a necessary condition for coordination. The learning technique used is X-classifier systems (XCSs).

We sketched a framework for adapting XCSs to learning behavior represented by Parts/whole Statecharts. We showed how to translate a Statechart transition into a classifier. The XCS environment includes the events and states of the Statechart, represented by data structures in the simulation, and the simulation maintains the data structures in a way consistent with the Statechart semantics of [6]. States and events in a transition are not just structurally related but also semantically related. We capture the semantics with feature vectors and consistency and agreement rules.

Two parts of our framework needing more work are learning structure and learning coordination (wholes). An attractive general approach is to zoom out to learn gross behavior of a group and zoom in to learn detailed behavior of its parts. Zooming in restricts the sources and targets of transitions and so alleviates part of the problem with new states. Finally, there are several automated translations [9] of Statecharts represented in automated tools into other representations, some close to our manual translations. We are investigating the translations to see whether we can adapt them or create similar tools.

8. ACKNOWLEDGMENTS

This work reported here is supported by General Dynamics under task ICA-05-03 (Tactical and Cooperative Behaviors).

9. REFERENCES

[1] Albus, J. *et a*l. *4D/RCS: A Reference Model Architecture for Unmanned Vehicle Systems*, Version 2.0, National Institute of Standards and Technology, Gaithersburg, MD, 2002.

[2] Barwise, J. *The Situation in Logic*. CLSI Publications, Stanford, CA, 1989.

[3] Chwe, M. *Strategic Reliability of Communication Networks*, http://www.chwe.net/michael/p.pdf, 1995.

[4] Fagin, R., Halpern, J.Y., Moses, Y., and Vardi, M.Y. *Reasoning About Knowledge*. The MIT Press, Cambridge, MA, 2003.

[5] Harel, D. Statecharts: A visual formalism for complex systems", *Science of Computer Programming*, *8, 3* (June 1987) 231-274.

[6] Harel, D., Pnueli, A., Schmidt, J., and Sherman, R.. On the formal semantics of Statecharts. In *Proceedings of the Symposium on Logic in Computer Science* (*LICS '87*) (Ithaca, New York, USA, June 22-25, 1987). IEEE Computer Society Press, Piscataway, NJ, 1987, 54-64.

[7] Huhns, M.N. and Stephens, L.M. Multiagent systems and societies of agents. In Weiss, G. (ed.), *Multiagent Systems*. MIT Press, Cambridge, MA, 2000, 79-120.

[8] Koestler, A. *The Ghost in the Machine*. Macmillan, New York, 1968.

[9] Mikk, E., Lakhnech, Y., Siegel, M., and Holzmann, G.J. Implementing Statecharts in PROMELA/SPIN," In *Proceedings of the 2nd IEEE Workshop on Industrial Strength Formal Specification Techniques* (Boca Raton, FL, Oct. 20-23, 1998). IEEE Computer Society Press, Piscataway, NJ, 1998, 90-101.

[10] Pazzi, L. "Extending Statecharts for Representing Parts and Wholes," In *Proceedings of the 23rd EUROMICRO Conference '97 New Frontiers of Information Technology* (Budapest, Hungary, Sept. 1-4 1997). IEEE Computer Society Press, Piscataway, NJ, 2000, 207-214.

[11] Wyatt, D. *Applying the XCS Learning Classifier System to Continuous-Valued Data-mining Problems*. Technical Report UWELCSG05-001, Learning Classifier Systems Group, University of the West of England, Bristol, UK, 2004.