MATHEMATICS OF COMPLEX ADAPTIVE SYSTEMS

JOHN R. CLYMER
Clymer & Associates, Boone, Iowa USA.

ABSTRACT

The mathematics of Complex Adaptive Systems (CAS) consists of mathematical linguistics, symbolic logic, and the physical laws of the system problem. These concepts are discussed in terms of a graphical simulation programming language designed to experiment with CAS that can occur at every level of creation: including social, ecological, molecular, and quantum. In order to understand CAS, the most important principle is understanding the interactions among entities. The understanding of these interactions is more important than understanding the entity features. For example, understanding the dynamics of individual vehicles is not enough to understand freeway traffic patterns. This is because freeway traffic depends on the decisions of drivers as well as vehicles' capabilities. For example, wave behavior occurs in freeway traffic when drivers are free to enter the freeway without restriction. This system behavior was found to be mitigated by controlling freeway entrance timing. Often, when knowledge of other entities are shared or are observable and the decision making rules used by entities to make decisions can evolve, emergent behaviors occur. Many examples of such complex systems are discussed, focusing on entities that are intelligent agents. A submarine classification sonar net and a vehicle traffic control network are presented as examples. A simulation programming system is described that allows experiments with the system behavior of intelligent agents to be performed.
Keywords: complex adaptive systems, intelligent agents, interacting concurrent processes.

1 INTRODUCTION

System behavior is described by the interaction of concurrent processes [1] where groups of processes describe the behavior of individual physical entities. Entities interact by communicating facts that logically suggest actions to be taken by other entities. Of course, entity capabilities are determined by the laws of physics.

A process can be thought of as a sequence of states that represent periods of time. As discussed later, these sequences are defined by transition rules that specify alternate state changes and associated state change event actions. States are usually associated with ongoing activities of entities as well as other periods of time. The flow of time for each process must be synchronized into a concurrent flow of time for all processes. A description or model of Complex Adaptive Systems (CAS), as presented in this article, requires mathematical linguistics (process description), symbolic logic (map facts into actions), and the laws of physics (physics of motion, electrodynamics, etc.)

If network entities are allowed to learn the rules that map facts into the choices among event actions and state activities, then an evolutionary, concurrent process may result describing the behavior of the network. If entities apply learned rules to decide event actions and state activities optimally, they may be called intelligent agents. As a guide to agent network evolution, consider Ross Ashby's law of requisite variety [2] that states that the degrees of freedom of the regulator processes must exceed the degrees of freedom of the processes being regulated. For example, a group of entities evolve in order to regulate (control) the effects of environmental processes. This law holds at all levels of the hierarchy of existence and guides natural evolution of entities to increase requisite variety.

Operational Evaluation Modeling (OpEM) for Context-Sensitive Systems (CSS) is a computer simulation tool OpEMCSS for networks of intelligent agents and environmental processes. It implements the science of complex systems using a logical, graphical language

© 2017 WIT Press, www.witpress.com
ISSN: 1755-7437 (paper format), ISSN: 1755-7445 (online), http://www.witpress.com/journals
DOI: 10.2495/DNE-V12-N3-377-384

www.

that defines the states and state transitions and includes tools that learn rules to regulate agent networks. OpEMCSS [3] allows the mathematics of complex systems to be executable, making experimentation with complex systems possible. Science is not science without experimentation.

2 MATHEMATICAL LINGUISTICS MODEL

In systems engineering, design documentation includes: (1) a collection of architectural diagrams representing physical entities and communication links among entities and (2) a collection of functional flow diagrams where functions are mapped to architectural entities. Before this design documentation is developed, the system problem must be specified in an executable form that allows for experimentation within the system's operational environment called the mission analysis and concept evaluation phases of the systems engineering process. In OpEMCSS, the operation of a system (network) of communicating agents in the system's operational environment is described by a collection of interacting concurrent processes called the operational view.

The operational view features a collection of interacting concurrent processes [1]. This view originally comes from mathematical linguistics using context-sensitive generative grammars as discussed in [3], Chapter 2. In order to understand this view one must learn to visualize strings (sequences of discrete states) representing the operation of an entity in the system. These strings are generated by regular grammars [4]. When the states of agents are synchronized in time, a context-sensitive grammar [4] is required. In other words, the generator rules of the language that define when each process can change state often depends on the state of other processes, making them context-sensitive.

There are several examples of context-sensitivity that will help understand from an operational perspective. Suppose that one process cannot continue (change state) until another process changes to a particular state. For example, the command and control process cannot start to process a target (an environmental entity) until the surveillance process detects it. The ship processes cannot detect the target until it is in the 'detectable' environmental state. This interaction is called 'synchronization.' There are many forms of synchronization that can occur.

For a second example, suppose one process is using a resource (a physical entity) required to perform the process function. A second process enters a state 'wait state' where it also requires the resource where it cannot change state until the resource is again free. This interaction is called 'resource contention.' Knowing the state transition rules for all concurrent processes allows the system scientist to visualize the concurrent operational state flow of the system.

Thinking about system operation using context-sensitive generator rules can be difficult. Further, if the operational view is to be executable, state variables such as resource counters (the number of physical entities currently available) must be provided to implement resource contention. OpEMCSS uses a graphical representation of concurrent process state flow. State variables are either global to all processes or local to a subset of processes. State variables allow detailed mathematical sub-models to be included in the process model. In cases where the laws of physics determine when state changes can occur, state variables allow detailed mathematical calculations to be done. For example, a detailed radar detection calculation (radar equation) could be used to decide when a target is detectable based on the target's process state and state variables. A ship model was very accurate in predicting ship system behavior given various environmental situations.

المنارة للاستشارات
www.

OpEMCSS simplifies a concurrent process, by allowing a graphical diagram to represent more than one process instance. This is called duplication. For example, consider a system that consists multiple detection sonars and multiple ships (surface ships and submarines). The sonars communicate to detect and identify the type of ship. The ships operate independently but interact with each sonar by going into and out of the detection range. The object of this experiment is to determine how to optimize the probability that each ship is identified correctly by type. The ships arrive and leave the scenario randomly so the number and type of ships in the scenario varies. For a complete description of the OpEMCSS software package see [3], appendices, and the sonar model [3], Chapter 8. The overview given in this article should allow us to understand applications involving CAS consisting of a network of intelligent agents [5].

Context-Sensitive Systems (CSS) are systems where decisions depend on the spatial and temporal context formed by interacting concurrent system processes with processes of the dynamic operational environment. CSS interactions result in very non-deterministic/non-linear system behavior: the same input does not necessarily produce the same output. Sometimes CSS interactions result in emergent behavior typical of what is known as a CAS. For example, a network of traffic light control systems [3], Chapter 9, where light control is context-sensitive can have an emergent behavior such that vehicle waiting time in the network is minimized. Self-synchronization of traffic light timing throughout the network, that results in minimum vehicle waiting time, is an emergent property of the entire network. More about this example is discussed subsequently.

3 EXAMPLE GRAPHICAL PROCESS MODEL

Figure 1 shows a sequential process diagram for a Fire Control System (FCS) as an example of an OpEMCSS graphical diagram. There are many other process models not shown with which the FCS process interacts. The states in the diagram are represented by circles with the name of the state shown beneath the circle. Alternative state changes out of the Designate FCS state are labeled Probability of Lockon (PLO) and Probability of Abort Lockon (PAB). The alternate action icon next to the state opens to reveal a mathematical equation to decide which alternate path to take. All OpEMCSS icons open to perform mathematical computations that allow the concurrent process model to be an executable computer program.

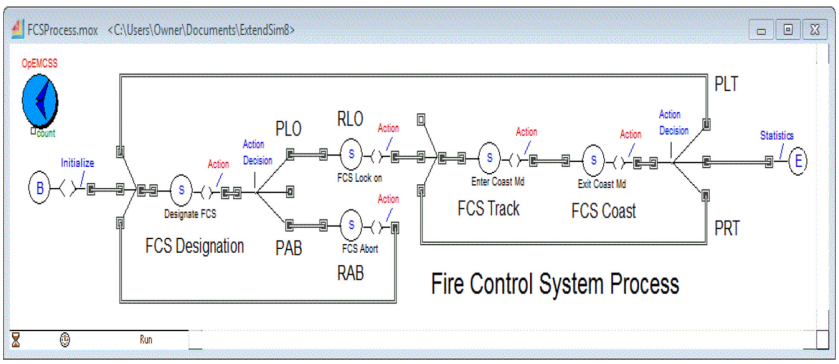

Figure 1: Fire control system process diagram.

المنارة للاستشارات
www.

4 AGENT FUNCTIONAL FLOW MODEL

Figure 2 shows a functional flow diagram of an intelligent agent consisting of five kinds of functions: (1) sensors, (2) transfer functions, (3) feature extractors, (4) classifier systems, and (5) evaluators.

4.1 Sensors and transfer functions

Sensors probe the environment and return raw data that describes what is out there. Signal processing transforms raw sensor data into refined data that measures various aspects of what is out there. Refined data is transformed into features (crisp or fuzzy sets) that encapsulate knowledge required for rules to make decisions.

For example, as discussed in [3], Chapter 8, for the sonar array model, when any ship moves through water, acoustic signals are generated and propagated omni-directionally. Each acoustic signal received by the sensor can be transformed into a set of features that measure some aspect of the received signal. Some of these measurements are the frequency, bandwidth, harmonic content, and signal stability. Feature 'frequency' is partitioned into about a dozen contiguous crisp sets where each crisp set is defined in terms of a range of frequencies. It is known that certain ranges of frequencies are associated with submarines and other ranges with surface ships. Therefore, if the collection of all frequencies detected is properly partitioned into crisp sets (feature facts), feature 'frequency,' along with other features, can be used to discriminate between surface ships and submarines.

4.2 Feature extractors and classifier systems

In the sonar example, a sonar expert was available to define the features. Given a scenario that no expert is available, the approach to the problem then becomes determining which signal data measurements relate more closely to the goals of the system. This problem can be solved by forming the refined data into crisp/fuzzy sets (feature facts) and using these feature facts to learn rules that make the best decisions. The OpEMCSS Classifier block was used to learn the rules that best classified surface ships and submarines correctly. This method was

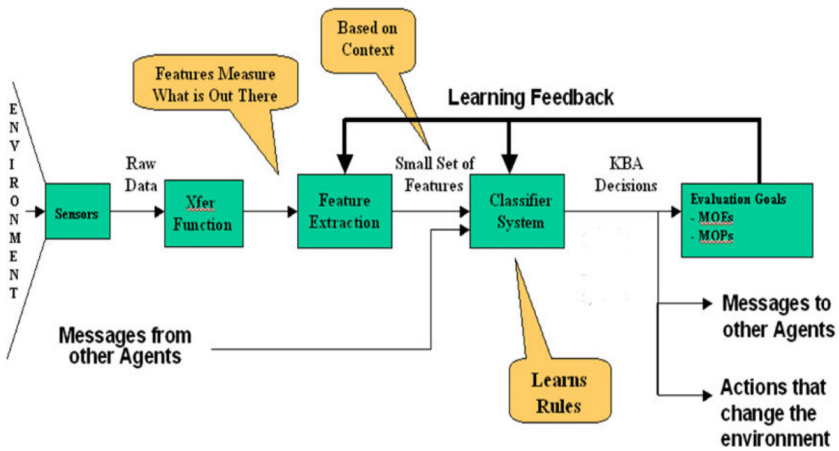

Figure 2: Agent functional flow diagram.

المنارة للاستشارات
www.

discussed in [3], Chapter 8, for the sonar array system model. Holland [6] was among the initial inspirations for the OpEMCSS Classifier block.

If the transformation from refined data to feature facts is not known, the Feature Extraction function learns to transform a large refined data space, where each dimension is a data measurement, into a more manageable and smaller feature space better suited to rule learning. Usually, feature space forms a closed manifold that is non-Euclidian (the features are not independent). Creating a feature extractor for a specific system design problem will probably require advance mathematics such as differential geometry. Transforming high-dimensional target description data into a small set of feature facts that provide correct target classification was accomplished using evolutionary algorithms [7].

4.3 Evaluator function

The reward criterion defines success and failure for decisions made by the OpEMCSS Feature Extractor block and the Classifier block. For example, consider the Prisoner's Dilemma game discussed in [3], Chapter 7, that does not require a Feature Extraction block.

Reward for Agent 1 is computed as per the rules of the game:

IF Agent1 = Collaborate && Agent2 = Collaborate Reward = 5;

IF Agent1 = Collaborate && Agent2 = Defects Reward= 0;

IF Agent1 = Defects && Agent2 = Collaborate Reward = 6;

IF Agent1 = Defect && Agent2 = Defect Reward = −3;

Rule learning for the Prisoner's Dilemma game always converges to both players collaborating. Axelrod [8] discusses many strategies for this game. Some of these are intricate ploys to trick your opponent into cooperating so you can defect; however, he shows that the best strategy is 'tit for tat,' where an agent punishes his opponent each time he defects by defecting next time and then returns to cooperating in a subsequent game. The learned classifier rules were the 'tit for tat' rules that were quickly learned.

In general, the system behavior achieved is always determined by the reward system devised. Therefore, the systems goals must be well understood and quantified by system behavior. Often, understanding the real quantified goals of the system is the initial object of the system design problem.

4.4 Learning feedback

Evaluations produced by the Evaluator function are feedback to the Classifier System function and the Feature Extraction function (this is a special block, user-defined mathematical algorithm). Both of these functions learn to optimize their operations in order to maximize the value of payoff received from the evaluator. Rule learning for the Classifier block is discussed in [3], Chapter 7. Learning for the Feature Extraction block is beyond the scope of this article and is a separate research problem for each CAS.

4.5 Complex behavior of intelligent agents

Some of the agent's decision actions may result in exerting partial control over the environment in a way that changes the environmental situation for other agents. The Prisoner's Dilemma model, discussed in [3], Chapter 7, is an example of two agents playing the Prisoner's Dilemma game. Each agent must decide on the basis of the outcomes of the last several games whether to cooperate or defect.

المنارة للاستشارات
www.

Physical agents may move and interact in the real world. For example, the world model, discussed in [3], Chapter 8, simulates 30 agents that move on a playing field and interact. The world model demonstrates how the agent motion and spatial interaction blocks are used; however, the world model is also a good example of a CAS. Each agent is modeled using three concurrent processes: agent motion, agent interaction, and agent group maintenance that are each duplicated for each agent. When an agent (Agent 1) comes into a specified interaction range 'IntRng' of another agent (Agent 2), Agent 1 adopts the velocity vector of Agent 2. Agent 1 also sends his new velocity vector to all other agents in a message. Any other agent that is within 'IntRng' of Agent 2 also adopts the new velocity vector. The overall behavior of this system begins with each agent randomly placed on the playing field with each agent having a random motion vector. The overall CAS behavior emerges to the situation where all agents are in a group with a common group motion vector.

The sonar sensor array modeled in [3], Chapter 8 is an example complex system where there are sonar sensor agents and environmental entities (i.e., surface ships and submarines). The sonar sensor agents have a fixed position in the physical world and do not move. However, the agents share their ship classifications with each other, applying a majority vote to decide the overall system classification. The ship entities, particularly the submarines, could have been modeled as intelligent agents that perceive their environment and adapt their behavior (change internal function) to minimize sonar sensor detection. The submarine agents would probably not communicate with each other (send output messages), however.

5 A SELF-ORGANIZING, CONTEXT-SENSITIVE SYSTEM

As an example of a CSS system having emergent behavior, a distributed vehicle traffic control network located in a large city is discussed in [3], Chapter 9. This traffic control network is an example of a System of Systems (SOS) where each system in the network independently provides specific services and each system can operate independently of the rest of the SOS. Additional services are provided through collaboration among the networked traffic control systems. Network Centric Operation (NCO) of related business units and combat system platforms are other examples of SOS that are currently of research interest [5].

Each major intersection has a vehicle traffic light controller to determine traffic light timing. In this system, each traffic light controller uses its sensor-obtained perceptions about incoming traffic flow to optimize light timing, thus minimizing local vehicle waiting time. The result of each traffic light controller adapting light timing to accommodate traffic flow coming from other intersections is to minimize the average waiting time in the entire network. Global minimization of traffic waiting time results as a consequence of the emergent behavior of this system, which is the self-synchronization of each traffic controller's light timing with all other controllers. Such self-synchronization results from the context-sensitive behavior of the network.

As light timing control in the overall traffic grid evolves in the way discussed above, a complex but definite pattern in network operation, north-south red to green transition times, emerges out of an initial random light pattern. The emergent behavior of the traffic grid cannot be explained through an understanding of each controller alone. Understanding only comes when we study the interactions of the controllers (the entire network) as they adapt their behaviors in response to sensor perceived information about incoming traffic flow, achieving self-synchronization of all traffic light controllers in the network.

The desired emergent behavior for the traffic control system was achieved by experimenting with feedback in the network. With no feedback, the traffic lights operate independently

المنارة للاستشارات
www.

and average vehicle waiting time is high. With very strong feedback, the traffic control network operation appears chaotic (little self-synchronization occurs) and the average vehicle waiting time is high. When the feedback is just right, operating on the edge of chaos, the emergent behavior (minimization of network waiting time) is observed. It is also interesting to note that the overall system never reaches steady state operation; indeed, the system seems to be in a constant state of flux as observed by the random appearance of the light timing control signals, even when the average vehicle waiting time is being minimized, implying emergent behavior.

A similar result occurs in a three-dimensional set of non-linear, simultaneous differential equations. Individual state variables appear random, but a graph of the three-dimensional point (x,y,z) follows a clear trajectory through space that never goes through the same point twice. In general, a single feature may appear chaotic, but several features can have a correlated path to optimal behavior. The Classifier block can follow the trajectory of the correlated path of features, even when they appear uncorrelated separately, to discover how to achieve optimal system behavior. It amazes me when it does it.

6 CONCLUSION

The behavior of a complex system that includes the interaction of multiple entities and the dynamic environment can be represented as communicating concurrent processes. Processes describe sequences of states where each state represents a period of time required to perform the action of the state. Process interactions include fact communication and forms of process synchronization usually determined by the physics of the problem (you can't process a target before you detect it). Logic is applied to facts to decide actions using rules. Rule learning allows communicating processes to evolve toward optimal system behavior, which may include regulatory actions on the environment. Such evolution can result in emergent behaviors for the whole system.

Operational Evaluation Modeling (OpEM) for CSS or OpEMCSS is a simulation library that works with ExtendSim, a product of Imagine That Inc. OpEMCSS and ExtendSim are available from [3] on a disk in the back cover of the book for immediate application to experimentation with CAS. Also provided in [3] is all information, including hands-on experiments, required to apply OpEMCSS to your complex system problem. OpEMCSS is designed to do complex system simulation and optimization. OpEMCSS includes: (1) blocks required to simulate interacting concurrent system processes within the dynamic operational environment; (2) blocks required to do evolutionary optimization of the system behavior; and (3) blocks required to simulate motion and spatial interactions among a set of physical entities in the system and its environment. OpEMCSS was created by the author to experiment with CAS and to greatly simplify creating a simulation program for system science study.

OpEMCSS blocks are connected in a network, as shown in Fig. 1, to simulate the execution of the concurrent threads of activities that comprise system and environmental behavior. System behavior can be optimized in terms of the number of each kind of resource and activity reaction times using the evolutionary algorithm block [7]. System management rules can be learned using the Classifier System block that optimizes system control decisions. Blocks are available to simulate motion and spatial interactions among a set of physical entities in the system and its environment. Often the position and velocity of system and environmental entities are critical to accurate description of system behavior in a dynamic operational environment.

المنارة للاستشارات
www.

OpEMCSS is a computer simulation program defined by graphical icons of process behavior, but allows for detailed mathematical models to whatever depth is required. Special blocks are included in an auxiliary library that allows a detailed mathematical model of entity physical operation to be included in an OpEMCSS simulation. Based on ExtendSim, OpEMCSS can be considered a general programming language with blocks provided to do discrete event simulation of interacting concurrent processes and entity behavior.

OpEMCSS is the culmination of 50 years of research, experimentation, and application to real world problems; most of them US DOD classified. The textbook [3] describes a methodology for the front end of systems engineering: mission analysis and concept evaluation where the real world problem is defined and top level requirements are specified. This paper focuses on the fundamentals of mathematical linguistics, symbolic logic, and the 'physics of the problem' required to do the front-end systems engineering of CAS.

REFERENCES

[1] Hoare, C.A.R., *Communicating Sequential Processes*, Prentice Hall: Englewood Cliffs, NJ, 1985.
[2] Ashby, W.R., *Introduction to Cybernetics*, Chapman & Hall: London, 1956.
[3] Clymer, J.R., *Simulation-Based Engineering of Complex Systems*, John Wiley & Sons Inc.: Hoboken, NJ, 2009.
[4] Hausser, R.R., *Foundations of Computational Linguistics: Human-Computer Communication in Natural Language*, 2nd edn., Springer: Berlin, 2001. https://doi.org/10.1007/978-3-662-04337-0
[5] Weiss, G. (ed.), *Multi-Agent Systems: A Modern Approach to Distributed Artificial Intelligence*, MIT Press: Cambridge, MA, 1999.
[6] Holland, J.H., *Hidden Order*, Addison-Wesley: Reading, MA, 1995.
[7] Fogel, D.B., *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*, IEEE press: Piscataway, NJ, 1995.
[8] Axelrod, R.M., *The Evolution of Cooperation*, Basic Books: New York, 1984.
[9] Haussler, D., Quantifying inductive bias: AI learning algorithms and valiant's learning framework, *Artificial Intelligence*, **36**, pp. 177–222, 1988. https://doi.org/10.1016/0004-3702(88)90002-1
[10] Westerdale, T.H., An approach to credit assignment in classifier systems, *complexity. Journal Santa Fe Institute*, **4**(2), pp.49–52, 1998.

المنارة للاستشارات
www.

© 2017. Notwithstanding the ProQuest Terms and Conditions, you may use this content in accordance with the associated terms available at https://www.witpress.com/journals/dne or in accordance with the terms at https://creativecommons.org/licenses/by/4.0/(the “License”), if applicable

المنارة للاستشارات

www.