Real-time Based Agent Architecture for Power Plant Control

Joel H. Van Sickel, Student Member, IEEE, and Kwang Y. Lee, Fellow, IEEE

Abstract—Building upon the work of the Multi-agent System (MAS) based Intelligent Heuristic Optimal Control System (MAS-IHOCS) a new Distributed Real-time Agent Framework with Time-warp (DRAFT) has been developed to enhance performance and development of multi-agent systems used to control real-time processes. Using distributed discrete event simulation techniques for optimistic simulation with time-warp speeds up the simulation of the multi-agent system. Time-warp capabilities are embedded directly into the agents, which are capable of switching between fast-as-possible simulation using time-warp or pseudo-real time simulation. This allows faster development of agent systems with more thorough testing and validation. Additionally, these critical systems must continue to operate safely and securely despite failure of agents. This requires graceful degradation of the multi-agent system. The DRAFT agents will be applied to a simple power plant for demonstration purposes of these qualities.

Index Terms - Graceful degradation, multi-agent system, optimistic simulation, power plant, real-time, time-warp.

I. INTRODUCTION

The area of multi-agent systems applied to power systems has seen an increase in popularity and acceptance. Most notable systems have been in development for a number of years. As progress continues to be made, areas of particular concern that can use more attention are integrating real-time scheduling with standard agent capabilities and appropriate methods for the simulation and validation of multi-agent systems working with real-time constraints. Also of importance are methodologies to decrease development time while enhancing quality and assuring reliability through extensive testing.

Distributed Real-time Agent Framework with Time-warp (DRAFT) agents were developed to address these issues while still maintaining many other useful aspects of multi-agent systems. The coupling of a real-time scheduling with standard agent decision making must be addressed in a manner that allows for the reliable performance required by real-time systems while allowing for the flexibility of agent computing.

Distributed discrete event simulation techniques for optimistic simulation, using time-warp, have been implemented directly into the architecture of DRAFT agents. This gives the agents the ability to be operated in a fast as possible simulation, or switch over to standard real-time operation. This allows development to proceed forward in a timely manner using fast as possible simulation to validate new concepts while allowing the small details of the system to be tested in a higher fidelity simulation that addresses the distributed and real-time nature of the applications.

Another important aspect that must be addressed in critical operations such as power plant control is graceful degradation. It is inevitable that an agent as some point in time will experience a failure. The system must be designed so that the system functions safely despite failures. This particular system degrades from a multi-agent system intelligent heuristic optimal control system (MAS-IHOCS) [1] down to a traditional power plant control system depending on the number of agent failures.

II. MULTI-AGENT SYSTEM

For power plant control, there are nine different types of agents that are used in this set up. These agents are heterogeneous because each of them serves a fundamentally different purpose than every other agent. However, each agent can be split into different types of categories as shown in Fig. 1. There are two main categories of agents, which use slightly different architectures.

**Hard Real-time Agents:** these agents must have the ability to interface directly with the real-time aspects of the power plant. They interact with other agents and the hardware, so they require a slightly more sophisticated architecture.

**Soft Real-time Agents:** these agents only interact with other agents, and while they have real time requirements, they are not as restrictive as those of the agents interfacing directly with hardware.

The soft real-time agents can then be generally categorized by the general purpose of the different agents.

**General Agents:** these agents have numerous functions and typically interact with all of the agents, humans, or a combination of both to do many different small tasks depending on the situation.

This work was supported in part by the National Science Foundation under grant ECCS 0801440.

J. H. Van Sickel is with the Department of Electrical Engineering, the Pennsylvania State University, University Park, PA 16802 (email: jhv107@psu.edu).

K. Y. Lee is with the Department of Electrical and Computer Engineering, Baylor University, Waco, TX 76798 (e-mail: Kwang_Y_Lee@baylor.edu).
Adaptation Agents: these agents serve the primary function of observing changes in the power plant and allowing the system to adapt to these changes.

Optimization Agents: these agents serve the primary function of optimizing the performance of the power plant for the current operating conditions.

A brief overview of each agent and its specific purpose follows:

Server Agent: The server agent serves the primary purpose of being an entry point for new agents and maintaining a list of all active agents and their functions. All agents have the ability to update their own lists, but having a server agent simplifies searching for other agents and handling broadcasts.

Interface Agent: The interface agent allows a human to interact with the multi-agent system. It provides data and parses information into agent messages and delivers them. It is primarily a terminal that can be outfitted with appropriate visualizations depending on the situation.

Fault Diagnosis Agent: This agent monitors operation of the power plant and diagnoses faults when anomalies are detected [2]. This could also be a cluster of agents and use more sophisticated diagnosis tools such as a multi-agent system in [3].

Identification Agent: The identification agent analyzes the operation data of the power plant to create or update models of the system. These can be dynamic models [2] or steady state models [4], which are then used by optimization agents to enhance the performance of the system. In previous literature, there were two separate identification agents, one for steady state and one for dynamic models. It has been found that one agent has the computation capabilities to satisfactorily do both tasks at the same time so they were combined into a single agent.

Reference Governor Agent: The reference governor agent takes the required unit load demand and generates set points and feedforward controls for the power plant [5,6]. These are found using a search algorithm with a steady state model of the system.

Gain Tuning Agent: The gain tuning agent uses forecasted load data to search for optimal gains for the feedback control system of the power plant [5]. In many ways it is similar to a reference governor, but it uses a dynamic model of the system which significantly increases the computation time required to obtain useful results.

Driver Agent: This agent is the primary means of interaction between the multi-agent system and power plant. It is responsible for taking agent data and formatting it for use with the power plant. This requires hard real-time capabilities. Often the driver agent will need to resample data streams to ensure synchronization.

Simulation Agent: When developing the multi-agent system or running it in simulation, the simulation agent represents the physical system, in this case, a power plant. There can be multiple simulation agents representing different aspects of the physical system, depending on what computational resources are available. While the simulation agent has standard agent capabilities, almost all of its computation power is directed toward simulation of the system.

Monitoring Agent: The monitoring agent collects and distributes data, providing another connection directly to the power plant. Other agents can request data from the monitoring agent, which will parse and deliver the data at whatever intervals and sampling rates are requested.

While these are the main types of agents, they do not necessarily require that nine different agents be used for a system. For small simple systems, different agents can be combined. All of these agents share a similar overall architecture and their operation is guided by the specific goals of the agents. A reference governor agent could be given the goals of an identification agent, and if it had the computation power, could carry out both tasks using the same processor.

Soft real-time agents can be combined together and hard real-time agents can be combined, but they should not cross over. It is possible to have a hard real-time agent carry out the
tasks of a soft real-time agent, but soft-real time agents typically have processor intensive tasks that would prevent the hard real-time agent from staying synchronized with the power plant.

III. REAL-TIME SCHEDULING

Something not often addressed when discussing agents is how they successfully interact in real-time environments. Many agent architectures being used do not inherently have a concept of time, or at least not one that drastically affects the agent’s decision making. For applications such as power plant control, this issue must always be at the forefront. While there are numerous sophisticated ways to make agents that act more intelligently, if they are doing this outside of temporal constraints, the results are essentially useless to real world applications with deadlines.

One of the primary challenges in scheduling different actions for agents is how to prioritize actions such as communication, which are necessary for the overall function of the multi-agent system, and real-time actions and tasks that are required for the power plant to operate successfully.

There are numerous techniques for doing multiple priority level real-time scheduling, but after analyzing the operation of the agents a simple approach was found that worked for this application. As mentioned previously, many of the tasks performed by the soft real-time agents are computational intensive. These agents are typically performing tasks on the time scale of minutes or more, while the communication and other small tasks take milliseconds.

This disparity of multiple orders of magnitude allows agent operations to be categorized into two separate time scales and a new approach was inspired by [7]. Large jobs are processed using earliest deadline first techniques (EDF), where the job with the earliest deadline is what is processed first. A fixed time slot is reserved for the smaller jobs. This is essentially a hybrid scheduling scheme that mixes fixed time slots with earliest deadline first techniques (EDF). With a large difference in time scales, the EDF scheduler can ignore all of the small actions required when scheduling large time-scale operations. This technique will degrade as the time scales approach to one another.

When an agent is idle, it can either continually reprocess within the fixed time slot, which may simply be checking for messages, or it can remain idle for a certain period of time before executing the agent brain, the logic that decides what goals the agent will pursue. An example showing how this is achieved can be seen in Fig. 2.

It is incredibly important to take this into account when developing the higher level agent logic that will assign agent actions and determine when they should be finished. This piece must work seamlessly with the agent brain of the system. If the agent brain schedules more tasks than is possible to achieve in a time period, the system will obviously fail. However, this can be avoided by properly developing the higher level logic for the agents.

This also means that the higher level logic should schedule events as soon as possible so the EDF algorithm has as much opportunity as possible to meet the real-time requirements. Another important fact is that a major assumption being made with this methodology is that it is alright to perform any operation earlier than its deadline once the agent logic enters it into the EDF scheduler.

IV. AGENT ARCHITECTURE

As mentioned in Section II, there are two types of agents, soft and hard real-time agents. From an architectural perspective, a hard real-time agent contains a soft-real time agent with additional functionality and communication capabilities. A DRAFT agent typically functions as shown in Fig. 3, which should be viewed with Fig. 2 in mind. One thing to note is that the agent logic block is where new jobs are inserted into the job queue.
However, the functionality of a hard real-time agent is achieved using the scheme shown in Fig. 4. This is possible through a different communication protocol that bypasses agent logic. Typically, all communication is filtered by the agent brain before it can be used for anything. This requires more processing because, firstly, the messages are generic and require more time to parse. Secondly, they must then be analyzed by the agent message handler before being acted upon.

To bypass these time lags, agents can establish direct connections that send data streams directly to where the data is required. This is shown in Figs. 5 and 6. This significantly increases the speed at which communication is possible and is how hard real-time agents should interact with other hard real-time agents and real world systems. This communication is primarily used for continuous streams of data, and is cycled many times more than the agent brain is.
V. DISTRIBUTED DISCRETE EVENT SIMULATION

A new fundamental change to the agents in this multi-agent system is that they all are embedded with discrete event simulation capabilities. These agents employ a scheme based on time warp and optimistic simulation [8].

Distributed simulations offer the advantage of faster simulations. Each processor involved in a distributed simulation is referred to as a logical process (LP). Each LP is responsible for simulating one piece of the full simulation and giving its data to the other LPS which require it for their own simulation purposes. The challenge is to keep the LPS synchronized. If the simulations process events are out of order, this is referred to as breaking the local causality constraint. The local causality constraint requires that actions be simulated in the order they are generated. Overcoming this issue is traditionally accomplished in one of two ways, conservative and optimistic simulations. Only optimistic simulation will be discussed here.

Optimistic simulation does not enforce the local causality constraint. Each LP is allowed to process at its own speed and ignore where other LPS are located in simulation time. It is called optimistic because each processor moves forward in time, optimistically assuming it has not violated the causality constraint. This will inevitably lead to the local causality constraint being broken at some point in simulation time.

To deal with this, a capability known as time warp is built into each LP, allowing them to step back, or time warp, to right before local causality was broken and proceed from there. This works because for local causality to be broken, it requires a time stamped message from another LP to be received. Once this happens, the LP that received the message too late now has the message it originally ignored, and can time warp back and proceed on, now having the correct information in hand. This LP must alert the other LPS as well that they must go back as well.

This is an over simplified explanation and can be compared to a group of people running together, and whenever they realize that someone has fallen behind, they run back to that person, and then continue on, however, they are not checking all of the time, which means the runner who fell behind may be able to catch up before they run back. This can be contrasted to a conservative simulation where the runners would never let themselves outpace from each other.

This optimistic scheme works because it is not unusual for an LP that falls a little behind to catch up later without ever having sent a message that would violate the local causality constraint. The challenge with an optimistic simulation is that time is wasted when recovering from violating the local causality constraint. If the recovery mechanisms are fast, it is not much of an issue, but even with fast recovery, if the LPS are so mismatched that they are continually having to time warp and resynchronize, the simulation could feasibly take longer than a single processor simulation. To prevent this, a mechanism can be used to slow down the faster LPS. One nice aspect in trying to achieve this is that if an LP is continually running to far ahead, it is aware of this and can choose to slow itself down in an attempt to lower the number of time warps being caused.

For this particular application, there is no lack of computing resources, but there is still a bandwidth requirement. This tips the scale in the direction of an optimistic simulation. Additionally, the look ahead issue is either impossible or extremely challenging in heterogeneous agent applications, but it is a simple matter to cause LPS moving too quickly to slow down. As the full system was fleshed out, other additional benefits of using an optimistic simulation system described in Fig. 7 became apparent as well:

- Time warp requires all previous events be remembered, this makes debugging drastically easier as all data on the system is already available (unless memory management requires some to be deleted).
- Time warp messages are easily embedded into already necessary agent messages.
- Time warp capabilities are easy to deactivate, allowing the same code to be used for a ‘fast as possible’ optimistic simulation as well as a soft real-time simulation.
VI. GRACEFUL DEGRADATION

Graceful degradation is where, as agents begin to fail, the system does not fail, but reverts to a less effective mode of operation. One of the easiest ways to achieve graceful degradation with a multi-agent system is to develop it based on standard techniques used before a multi-agent system was put in place. The multi-agent system should be designed so that as it fails, it turns control back over to the system that was in place before the multi-agent system.

For power plant control, this is done by implementing the agents so that their control techniques are based on standard power plant control techniques of PID feedback controllers and a master control unit that provides set points and possible feed forward control actions. This allows the system to be easily (in a relative sense, it is still time consuming and expensive) implemented in a typical power plant without large modifications, and will also allow graceful degradation.

This can be seen by observing the specific functions of the agents. A failure in the server, fault detection, or interface agent does not immediately cause a failure in the power plant operation. Failure of the identification agent or monitoring agent takes away the ability for the system to adapt. Failure of the gain tuner causes the gains to no longer be optimized. These failures also do not cause any immediate problems. If the reference governor or driver agent fails, however, the system will no longer get required signals for plant operation. This is where the standard master control unit must still be attached to the power plant, so it can switch over modes of operation and continue to function. Now that all of the agent functionality is gone, the system is just the same as it was before. This is the ideal situation. Shown in Fig. 8 is an example of switchover from reference governor failure.

Fig. 8 shows the MAS operating correctly on the power plant used in [4]. At 140 minutes the reference governor agent stops functioning, and the system transfers over to baseline operation. The result is that the unit load demand is not tracked quite as quickly, but the biggest disparity is that the overall efficiency is worsened as 1.43% more fuel is required when the multi-agent system is no longer functioning fully.

VII. SUMMARY

A basic overview of a new architecture for a multi-agent system used to control power plants has been presented, focusing primarily on adding discrete simulation capabilities to agents and combining agent logic with real-time scheduling. Also addressed is the issue of graceful degradation under agent failure scenarios. This architecture will be more fully fleshed out and implemented on a full scale power plant, but shows promise for combining agent flexibility with the actual constraints of real world operation.

VIII. REFERENCES


IX. BIOGRAPHIES

Joel H. Van Sickel received his B.S. degree in Electrical Engineering from Grove City College in 2005. He is currently pursuing his Ph.D. in Electrical engineering at the Pennsylvania State University. His interests include distributed artificial intelligence, distributed simulation, intelligent control, power generation, multi-agent systems and optimization techniques.

Kwang Y. Lee received his B.S. degree in Electrical Engineering from Seoul National University, Korea, in 1964, M.S. degree in Electrical Engineering from North Dakota State University, Fargo, in 1968, and Ph.D. degree in System Science from Michigan State University, East Lansing, in 1971. He has been with Michigan State, Oregon State, Univ. of Houston, the Pennsylvania State University, and Baylor University where he is currently a Professor and Chair of Electrical and Computer Engineering. His interests include power system control, operation, planning, and intelligent system applications to power systems. Dr. Lee is a Fellow of IEEE, Associate Editor of IEEE Transactions on Neural Networks, and Editor of IEEE Transactions on Energy Conversion. He is also a registered Professional Engineer.