Proceedings of ASME 2011 5th International Conference on Energy Sustainability & 9th Fuel Cell Science, Engineering and Technology Conference
ESFuelCell2011
August 7-10, 2011, Washington, DC, USA

ESFuelCell2011-54619

AN INTELLIGENT SENSOR FRAMEWORK FOR THE POWER GRID

Bora Akyol, Jereme Haack, Cody Tews, Brandon Carpenter, Anand Kulkarni, Philip Craig
Pacific Northwest National Laboratory
Richland, WA 99352
Email: bora@pnl.gov

ABSTRACT

The number of sensors connected to the electric power system is expected to grow by several orders of magnitude by 2020. However, the information networks which will transmit and analyze the resulting data are ill-equipped to handle the resulting volume with reliable real-time delivery. Without the ability to manage and use this data, deploying sensors such as phasor measurement units in the transmission system and smart meters in the distribution system will not result in the desired improvements in the power grid. The ability to exploit the massive data being generated by new sensors would allow for more efficient flow of power and increased survivability of the grid. Additionally, the power systems of today are not capable of managing two-way power flow to accommodate distributed generation capabilities due to concerns about system stability and lack of system flexibility.

The research that we are performing creates a framework to add "intelligence" to the sensors and actuators being used today in the electric power system. Sensors that use our framework will be capable of sharing information through the various layers of the electric power system to enable two-way information flow to help facilitate integration of distributed resources. Several techniques are considered including use of peer-to-peer communication as well as distributed agents.

Specifically, we will have software agents operating on systems with differing levels of computing power. The agents will cooperate to bring computation closer to the data. The types of computation considered are control decisions, data analysis, and demand/response.

When paired with distributed autonomous controllers, the sensors form the basis of an information system that supports deployment of both micro-grids and islanding. Our efforts in the area of developing the next generation information infrastructure for sensors in the power grid form the basis of a broader strategy that enables better integration of distributed generation, distribution automation systems and decentralized control (micro-grids).

NOMENCLATURE

Agent — An agent is defined to be a software entity that is capable of accomplishing a given task on its own. An agent can be mobile or fixed.

Software Framework — A software framework provides the platform upon which the agents described in the text execute and accomplish tasks. The framework hosts the agents.

INTRODUCTION

Significant portion of the smarts in the *smart grid* efforts to modernize the electric power system come from application of many sensors to optimize asset efficiencies. The number of sensors connected to the electric power system is expected to increase by several orders of magnitude by 2020 [1]. These sensors include (but are not limited to) synchrophasor sensors deployed throughout the transmission system as well as smart meters, responsive home and water heaters in the distribution system.

The research presented in this paper creates a distributed agent-based framework to add "intelligence" to the sensors and controllers being used in the electric power system today. These intelligent sensors are capable of sharing information through the various layers of the electric power system to enable brand new

applications including two-way power flow to help facilitate integration of distributed generation. Finally, when paired with distributed autonomous controllers, the sensors can form the basis of an intelligent system that supports both micro-grids and islanding. While not discussed in this paper, our intelligent sensors are capable of adjusting measurement methods to the sensor environment and are able to perform measurements in multiple dimensions such as information networks and electric power system simultaneously.

Our research is significant because it lays the software platform groundwork for distributed operation and control of the electric power system especially at the distribution layer where end customers are being served. The intelligent sensors and the distributed control paradigms enabled by this intelligent sensor framework improve system reliability and support integration of distributed generation.

Our objective is to develop an agent software framework that adds intelligence to networked sensors in the electric power system in order to:

meet real-time data processing requirements;
adjust automatically the resolution and frequency of data being sent across the sensor network based on current state of the systems being observed by the sensor;
increase the scalability of the information networks;
explore new ways of correlating data from multiple domains;
improve control system response times by locating intelligence closer to the control points;
create the information channels to enable two-way power flows to speed up the integration of distributed generation in the power system, to allow decentralized control to support micro-grids and islanding.

The intelligent agent-based sensors we envision are capable of operating autonomously without requiring intervention from a centralized control point. They adapt and tailor their measurement methods based on environmental conditions and perform multiple types of measurements belonging to different domains to enable localized autonomous controls while also communicating this information back to a system operations center. Alternately, measurements from multiple single-purpose sensors may be combined by an intelligent sensor aggregator to support legacy sensors that already exist in the power grid.

This manuscript presents the results of an ongoing research project with the goal of introducing our work to a broader community and getting feedback. The rest of this manuscript is organized into three main components. First, we describe potential use cases and applications of agent-based intelligent sensors in the future power grid. Second, we summarize our software framework paying particular attention to requirements that are specific to the power grid. Third, we present our distributed agent simulation environment and our results. We present our future work directions in the conclusion.

POTENTIAL APPLICATIONS OF INTELLIGENT SENSORS IN THE FUTURE POWER GRID

In this section we discuss potential applications of agent-based, intelligent sensors in the future power grid through use cases that we developed as part of our requirements analysis. The use cases listed here are not an exhaustive list, but are included in this manuscript to illustrate what is possible when we distribute intelligence instead of locating it at a central point. The use cases that we selected for this manuscript are:

1. Distributed Generation as Forecastable Resource
2. Load sequencing by use of agents
3. Goal-based demand-response using distributed agents in the power system

Use Case 1: Distributed Generation as a Forecastable Resource

There is uncertainty around integration of distributed resources such as rooftop solar panels, end-customer wind turbines, battery storage and plug-in hybrid electric vehicles (PHEVs). An electricity generator plugged into the electric power system at the distribution layer is treated as "negative" load. This means that it is not actively accounted for and is not integrated into the electricity operations planning like a bulk power generator. Specifically, even when ample capacity is available from distributed resources in the distribution system, the system is unable to accommodate this capacity by flowing power from the distribution system back to the transmission system and to a different location where the power may be needed.

This use case is about turning a set of distributed resources connected (electrically) to the distribution system into aggregated forecastable resources. We can accomplish this type of resource integration by means of an agent based software framework. The agent framework has three components:

1. A forecaster/historian agent located at the distribution substation
2. Resource availability agents that gather a list of distributed resources and monitor their status in real time
3. Resource integration agents that are responsible for collecting resource generation information. The collected information is then made available to the forecaster located at the distribution substation

The (distributed resource) forecaster collects information generated by the agents and is capable of generating forecasts looking ahead at 60 minute, 360 minutes and 24 hours intervals. These forecasts can be made available to the utility system operations center and they can be treated similar to a forecast coming from a bulk power generator.

Use Case 2: Sequencing Load in the Distribution System

Consumer loads are largely a resource out of control of the utility. A popular demand/response technology calls for two way

communication between the utilities and consumers, allowing the utility to shed certain loads during peak loads. One side effect of this type of technology is that the loads get synchronized and the resulting power draw on the feeders has large peaks and valleys. This causes almost the opposite effect of the desired resource management goals.

A potentially more efficient and less intrusive system would be to sequence certain types of consumer base loads, providing a more stable equivalent load to the sub-transmission system.

Imagine a neighborhood in which each of the water heaters has an intelligent controller allowing for intra-feeder communication. Through auto-negotiation the water heaters decide that they have peak usage in the morning and rather than all turning on at the same time they will use a time-slicing algorithm to heat water one (or two) at a time. Additionally they could be aware of typical usage time and pre-heat in a similar fashion. Of course if the temperature dropped below some threshold a device could drop out of the time-slicing mode; however the general use would be coordinated.

Another potential consumer load of interest would be heat and/or cooling systems. It would be optimal for a consumer's end service to be non differentiable from a non-sequenced world.

A substation could receive knowledge of this time-slicing behavior and use it for its own planning needs. An agent based framework provides the capabilities expected:

1. Consumer level distributed agent technology which can describe load characteristics (time of day, duration, frequency, seasonal adjustment, etc.)
2. Sequencing / voting (based on load characteristics develop an optimal slicing scheme)
3. Automatic grouping (based on the number of sensors, automatically break up into "smart" groups)

Use Case 3: Goal-based demand-response

There are two types of demand response schemes currently employed in the electric power system. The traditional demand response schemes are based on direct load control where a utility by means of a communication channel triggers a responsive asset to turn on and off. A second type of demand response (while not prevalent in the field) uses a price signal to elicit customer behavior that results in reduction of load through voluntary means. Note that while direct load control is deterministic (mostly), price-signal based demand response is non-deterministic. The Olympic Peninsula project showed that it is possible to elicit load reduction better than direct load control using price-signal based demand response [2].

This use case utilizes our distributed agent technology for a collaborative demand response scheme with interacting agents that work together to achieve a demand management goal and confirm the amount of load reduction to the rest of the utility network thereby alleviating the non-deterministic nature of price based demand response. For example, we can define a (demand management) Zone Controller as an entity that sets demand reduction goals for the zones that are under its control/influence. The zone controller assigns a load reduction goal to its subsidiaries and this zone reduction goal flows through the system from the 1st tier (which could be interpreted as a region controlled by a balancing authority) to 2nd tier zones (e.g. a utility) and all the way down to individual distribution substations . The agent framework can be extended to equipment located at customer premises as well. At each tier, the zone controllers are implemented as "software agents". Specifically, they collaborate with each other, exchange information (securely) and iterate until desired goals are reached.

The amount of demand response achieved is collected and communicated. Therefore, the zone controller does not need to rely on metering observations to see whether the demand response scheme is working. Agents for each household could have different settings for priorities (maintain power level vs. take advantage of price). Aggregate result of the interaction of those agents would be the response for the "neighborhood".

A SOFTWARE FRAMEWORK FOR INTELLIGENT SENSORS FOR THE POWER GRID

Our research focuses on creating a usable framework for adding intelligent software agents to both existing and new devices that are deployed in the power grid. In this section, we will summarize the requirements we have discovered through use case analysis and from discussions with power system experts. We then present our design choices for the agent framework. We note that the research we present here is still ongoing. One of our goals in presenting our results and analysis as soon as possible is to gather feedback to help shape the future years of our work.

Requirements for software sensors for the power grid

Our research team has analyzed a number of use cases from power transmission, and distribution systems to derive requirements for the sensor platform. Note that it may not be feasible to integrate the rich set of functionality to every sensor, and control element used in the power system. Additionally, due to long equipment replacement cycles in the power industry, it may not be feasible to go ahead and replace old equipment immediately. In order to accelerate the deployment of intelligent sensors in the power grid, we propose to use a proxy device that exists in between legacy sensors and control applications and provides most of the enhanced functionality described in this manuscript. This device will be referred to as an "intelligent sensor aggregator (ISAg)."

Hardware Requirements: In order to allow maximum flexibility for the software framework, our hardware requirements are not very demanding. However, there will still be instances of hardware such as an 8-bit microcontroller that is not

able to run the framework. For devices that are not capable of accommodating the agents, as we discussed in the previous section, an *ISAg* device can act as a proxy platform and still enable the functionality that is required.

The platform hardware requirements can be separated into central processing unit (CPU), memory, storage and inputs/outputs. For CPUs, 32 bit addressing, and a memory management unit (MMU) that supports virtual memory, paging and process memory segmentation are highly desirable. Note that the MMU requirement is not very severe today since in the last 10 years, embedded microprocessors from many CPU families including ARM, PPC, x86 have all gained an MMU. The primary benefit of the MMU in embedded computing is to be able to support memory isolation and segmentation between different processes running on the CPU. It enables different agents running on the system to be isolated from each other and improve the cyber security of the platform.

The memory (RAM) located on a sensor used in the power system is a lot less than what is now common in consumer applications. It is common to see 8MB or less memory on older equipment whereas modern equipment like a phasor data concentrator (similar in functionality to an ISAg) may have upwards of 2GB of memory. For the purpose of this project, we will target devices with 32MB or more memory while paying attention to detail to ensure that the platform can be sized down to lower memory sizes.

The devices we will be targeting in this project are required to have at least 128MB non-volatile storage. Typically in control systems, flash memory is used and the operating system must be aware of wear associated with flash memory when information is written to the same sectors repeatedly.

A sensor may have many inputs and outputs. The sensor platform we are designing here has no specific requirements on inputs and outputs. An ISAg may have many serial and networked inputs. Since the sensors are assumed to be networked, they have at least one wired or wireless network interface.

Operating System Requirements: The capabilities developed in this project will be operating system agnostic. Given the memory, storage and CPU capabilities of devices that we typically see in the power system, the operating system we choose must be capable of running on devices with a memory of 32MB and reserve no less than 16MB of memory for sensor agent framework for agents.

Diagnostic and Manageability Requirements: One of the advantages of using agents in our framework is the ease of diagnostics and management. Agents can be dispatched to collect data as they move through the system and then carry that data back with them to a management station. The platform must be able to manage the activities of all hosted agents. The management may include stopping, restarting or *banning* an agent deployed in the system.

Information Sharing Requirements: To be able to share information is a key tenet of any distributed system. As demonstrated by the use cases we described previously, the agents in the power system are capable of accomplishing tasks that would require tremendeous amount of collaboration if one tries to implement the same capability using a centralized system. However, we also need to be careful about how much information is shared and where in order to prevent an information overload and keep information as secure as possible. Therefore, we will divide the operational area of the sensors to designated sensor collaboration regions (SCRs). Furthermore, the sensors may be divided into sub-groups referred to as "sensor task group" (STG) that are task based. For example, all sensors collaborating on accomplishing load sequencing may belong to a "load sequencing" STG[1]. The key requirement is for sensors in an STG to be able to freely exchange information. Moreover, there may be information that needs to be exchanged (such as management information) for all the sensors in an SCR. The agent-based software framework that we are developing must be able to accommodate many-to-many secure and reliable information exchange.

Agent Software Framework Runtime Environment Requirements: The intelligent sensor framework presented in this manuscript supports software agents. Software agents are entities that can either be stationary or roam between sensors in the system. Agents are task driven and can be used to accomplish a task that involves many sensors. The following are the requirements for our software framework:

1. Agents run on multiple CPU architectures as long as agent resource requirements are met. We must provide the agents with a runtime environment independent of the underlying CPU architecture.
2. Agent framework must be able to determine whether it has the capabilities to support a particular agent. In essence, the platform and the agent enter into a binding contract when a platform hosts an agent.
3. Agent framework must be able to determine if the agent requesting "hosting" is authorized to execute on that platform.
4. Agent framework must be able to determine if the agent code has been tampered with.
5. Agent Framework must allow agents to store information on-board.
6. Agent Framework must provide access to information collection and dissemination services for the agent.

[1] We should note here that the sensor collaboration regions and sensor task groups can be easily implemented using Internet Protocol (IP) multicast protocols.

7. Agent Framework must be able to guarantee (stochastically) a requested slice of its resources such as memory, storage, processor cycles to an agent.
8. Agent Framework must be able to detect and terminate an agent that is not behaving according to its agent/platform contract.
9. Agent Framework must be able to collect and transmit an inventory of agents hosted by the platform.
10. Agent Framework must implement a liveness check for each agent hosted on the platform in order to detect and terminate malfunctioning agents.
11. Agent Framework must provide a protected environment for each agent. This could include memory and storage protection.

Configuration, Change and Agent Update Requirements: Another significant benefit of an agent-based software paradigm is the ease with which configuration, change and agent software updates can be performed. For example, if smart meter data collection is implemented as an agent, we can update the metrology agent without needing to perform a costly and time consuming monolithic firmware upgrade for the entire platform image. A new version of an agent can simply be dispatched by means of *an update agent* that moves through the system. An update agent can also perform configuration and change management duties. The main requirement for the agent framework is a well-defined interface for agents such that an agent can be upgraded without needing a complete software update for the entire platform.

Security and Robustness Requirements: The sensors and controllers in the power system are networked devices. Our intelligent agents require ubiquitous information sharing capabilities to be able to perform their tasks collaboratively. Any system that is comprised of networked devices must meet the following basic requirements:

1. The networked devices must be able to handle incoming network traffic at the full data rate supported by its interfaces. For example, a sensor that has a 100Mbps Ethernet interface must be able to receive packets at that full rate indefinitely and not crash or hang. Note that especially for platforms with limited processing power; this requirement may be met by means of hardware support.
2. The device hardware must provide interfaces to the agent framework to accommodate physical tamper detection warning systems and respond to an event generated by the tamper detection circuitry.
3. Agent Framework must validate the software as part of the boot process.
4. Agent Framework must periodically validate the software to detect tampered executables if the software framework resides in modifiable storage.
5. Agent Framework needs the means to lock certain memory locations as read-only once the boot process is completed. This may include all cryptographic and authentication libraries.
6. Agent Framework must be able to detect resource exhaustion (either malicious or due to defective software) and take corrective action.
7. For battery operated platforms, platform must be able to detect behavior that causes battery exhaustion and take corrective action.
8. Agent Framework must be resilient against misfiring interrupts. The platform must be able to mask a misfiring interrupt.
9. Agent Framework must offer memory protection to hosted agents and PSMs.

In addition to the basic requirements, the agent framework must be able to meet authentication, authorization and accounting (AAA) requirements as discussed in [3] as well as requirements for communication protocols security. Additional cyber security requirements may also be found at "Guidelines for Smart Grid Cyber Security" published by NIST [4].

INTELLIGENT SENSOR SOFTWARE FRAMEWORK DESIGN

The intelligent sensor software framework needs to meet the requirements that we discussed in the previous section. An overview of the sensor framework is given in Figure 1. The sensors in our environment are networked devices with digital and analog sensing (and control) interfaces. The operating system shown in a dashed outline is an optional component. Note that there are runtime environments that can run directly on hardware without the need of an operating system [5]. The sensor framework shown as a component above the operating system is the *main* runtime environment for the agents discussed in this paper.

The internals of the sensor framework are shown in Figure 2. At the base layer, we show either a direct hardware interface that allows the sensor framework to run on top of hardware; or an operating system interface that allows the sensor framework as a module inside an operating system. The agent network communication layer inside the framework provides a network communication interface including the necessary cyber security functions for data transport. The agent discovery, cooperation and collaboration layer allows the agents to discover other agents and devices in their environment. The agents are then capable of exchanging information securely and cooperating to accomplish tasks as we discussed previously in the text. The agent authentication and authorization layer validates that an agent is authorized to execute on the sensor and that it is not tampered with during transit. The authentication and authorization of the

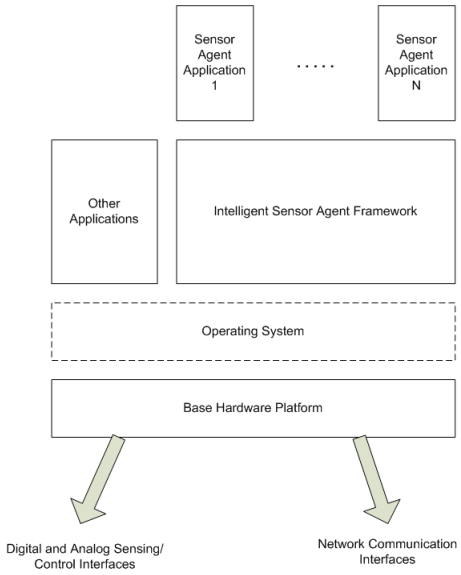

FIGURE 1. Intelligent Sensor Framework Overview

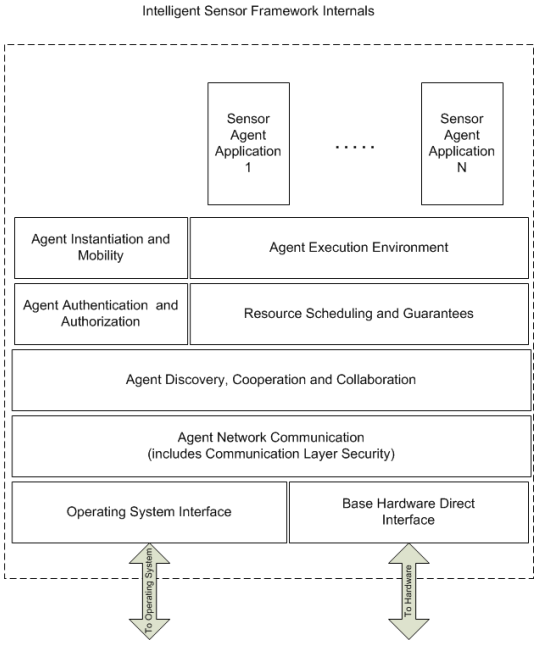

FIGURE 2. Intelligent Sensor Framework Internals

agent can not be completed without consulting with the resource scheduling and guarantees layer. The resource scheduling layer determines whether the platform has sufficient resources to accommodate this agent. The agent instantiation and mobility layer supports movement of agents between devices. Finally, the agent execution environment provides all other services that allow an agent to accomplish its tasks.

INTELLIGENT SENSOR SOFTWARE SELECTION

We have considered Java, Python and Lua for implementing our software framework. Python [6] is a general purpose programming language similar in usability to C/C++ and enjoys a broad user base. Python is supported by many architectures. Lua [7] is a powerful, lightweight, fast, embeddable scripting language. However it is not object oriented. One of the main requirements of platform is ability to load/unload modules. Also, python is better equipped in terms of functionality, tools, and reference manuals [8]. Using Lua would put a lot of burden on programmers to achieve this functionality. There are many versions of Java virtual machines (JVM) which have been modified for embedded systems. The main advantages of using Java are: portability, software reuse, built-in safety and security mechanisms, and memory management. Some of these advantages also apply to Python and Lua. Python needs a underlying operating system to operate on sensor node. It supports modular design and we can write applications which dynamically load/unload modules from the node. Both Java and Python provide a garbage collection mechanism. As for protection and authentication of modules, programmer can write interfaces to implement the desired functionality. Python provides better memory protection than C since Python does not use pointers or access memory in a direct way. The downside is that it is interpreted and therefore slower.

One of the JVMs we considered for our purpose is Squawk VM [5]. The Squawk VM, is an open-source platform for wireless sensors. It is certified on the Java Micro Edition Information Module Profile (CLDC 1.1 IMP 1.0) [9] and provides developers with the benefit of standardized Java language. Squawk is mostly written in Java and can run without an OS directly on hardware. This is one of the major advantages of squawk compared to other platforms. This capability makes the operating system optional in our framework. The architecture of the Squawk VM was inspired in part by the Squeak [10] and Klein VM [11] architectures.

Squawk can be used to compile and optimize code VM components, other Java components, and bytecodes ahead of time (AOT). Squawk is designed to run multiple applications using an application isolation mechanism such that each application, called an ”isolate,” maintains its own copy of its mutable or modifiable system state. This will greatly help us develop agents, which can be dynamically loaded/unloaded from sensor platform. Each application is represented by a Java object, which is an instance of class Isolate. The synchronization of resources is tracked on a per-isolated-application basis.

Squawk also provides thread synchronization. Threads in one isolated application are managed as a group without affecting threads in another isolate [12]. The Isolate instance can be used to query status of an application associated with it, and even start, stop, pause or resume application. This provides us

with complete control of Isolates, which is highly desired for security and one of the major requirements. Another capability that we want from our platform is the ability to transport agents from one node to other. In case of Squawk, Isolates will have to be loaded/unloaded and transported. In Squawk, the entire state of an application (treated as isolate) is stored as java objects which can be serialized to disk/stream. Because each Isolate can be serialized onto a stream, other nodes can read that stream into another Squawk VM to immediately reconstitute the isolate [13]. This migration of isolates enables us to implement roaming agents easily.

Apart from selecting a platform which supports dynamic loading of modules, an agent deployment tool enables us to transport these modules (or isolates) between different nodes. The agent deployment tool provides a runtime environment which is used to actually run the agent application. This runtime environment, implemented in Java, ensures transport of messages about agents, registration and de-registration of agents in the community, and also lookup for services provided by agents themselves. FIPA specifications addresses several aspects of agent system development [14].

Some agent and communication systems that we considered for our architecture are JADE, JINI, and Cougaar.

Jini was developed and introduced by Sun Microsystems in 1999 [15]. Jini is a service oriented architecture for networks services that addresses problems specific to distributed systems. Jini uses a lookup service to broker communication between the client and services. This appears to be a centralized model (though the communication between client and service can be seen as decentralized). This kind of architecture does not scale well for large systems.

The other agent system that we considered, Cougaar [16] is also Java based. It is used for highly scalable distributed agent-based applications. It is the product of multi-year DARPA research project to develop an open-source agent based architecture. One of the major disadvantages of cougaar is that it is not FIPA compliant.

JADE (Java Agent Development Framework) is a software framework fully implemented in Java [17]. It provides a set of Java classes that allow a developer to build a FIPA-compliant multi-agent system quiet easily. Jade also provides a set of graphical tools that help user to implement a multi-agent system. The configuration can even be changed at run-time by moving agents from one machine to another when required. An add-on to JADE called LEAP replaces some part of JADE kernel creating a modified environment called JADE-LEAP which allows the implementation of agents in mobile devices with limited resources.

The synergy between JADE and LEAP allows us to obtain a FIPA compliant agent platform with reduced footprint and compatibility with mobile Java environments down to J2ME-CLDC MIDP 1.0. For now, we have decided to rely on JADE-LEAP agent system for communication/transportation of agents

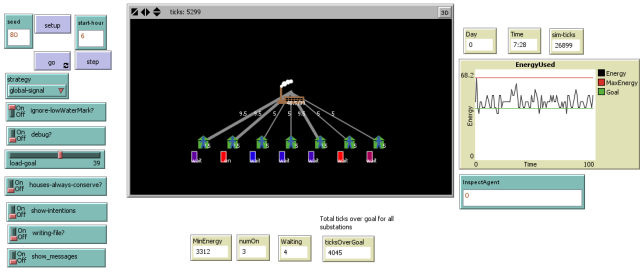

FIGURE 3. NetLogo simulation showing agents and controls.

between nodes. During development, if we are not satisfied with the performance of JADE-LEAP, then we will re-evaluate the agent systems and other options.

DISTRIBUTED AGENT SIMULATION ENVIRONMENT & RESULTS

In order to explore our concepts we first built a conceptual demo in NetLogo [18]. This allowed us to concentrate on our ideas without worrying about implementation at the same time. NetLogo was chosen for its ease of use, informative documentation, and graphical nature which makes it useful for demonstration purposes. NetLogo also makes it simple to add graphs for displaying data about a simulation in progress and aids in understanding what the agents are doing at any given tick. Figure 3 shows the NetLogo simulation where components of the energy delivery system were modeled as agents.

Simulation Terminology

Ticks: In NetLogo a ”tick” defines a ”turn” in the simulation where every agent gets a chance to take its action based on its current state. For this model, ticks have been mapped to seconds.

Substation: A substation is responsible for delivering energy to houses downstream. It receives energy goals from upstream which it must meet to ensure smoother energy use throughout the grid. When it detects that energy use will be above the goal, it can send conserve signals to its houses asking them to reduce consumption.

House: A house receives its energy from a Substation. A house can have several appliances (HVAC, refrigerator, etc.) as well as a *base load* which takes into account all other uses (lighting, television, etc). In this model, only water heaters are modeled, but the agent based setup eases the addition of other appliances.

Water heater: A water heater draws power from its house. It can be in one of two states: waiting or heating. While waiting it is monitoring the energy of the water to make sure it is at the goal. If water energy drops, then it enters the ”heating” state where it heats the water until it reaches the goal. Once the energy of the tank is at the goal, it goes back to the ”waiting” state. Water heaters also have a ”low water mark” which indicates that

tank energy is critically low. In the GUI, the tanks energy is represented by its color from blue (low energy) to red (high energy).

Residents: The inhabitants of the house are represented by the Resident agent. The function of this agent is to "consume" the resources of the house which can cause other agents to take actions. For instance, each resident has a schedule for when they will take a shower which has the effect of reducing the energy in the water heaters tank.

Distribution Lines: These transfer energy from the substation to the houses. In this model they serve no function other than as a display for showing how much energy is being used represented by line thickness.

IgnoreLowWaterMark: By default, if a water heater is under its LowWaterMark, its house will ignore a conserve request.

EnergyUseToken: A house possessing a token can draw its required load from the substation.

Simulation Description

For every tick in the model, there is a sequence of agent actions that happen. First, water heaters in a waiting state enter a heating state if their tank energy is below goal. Water heaters in a heating state enter a waiting state if the tank is at goal. Next, houses estimate their energy usage based on the state of their water heaters and their base energy usage. Substations receive these estimates and compare them against their load goal. If the total is over load goal, they enter a conserve state. Substations in a conserve state send a conserve signal to their houses and enter a monitor state if their estimate is under goal or a a conserve timeout has elapsed. A house receiving a conserve signal polls its appliances as to their ability to conserve. Water heaters willing to conserve are flipped back to their wait state.

There are three different strategies employed by the agents in our model to deal with energy usage and attempts to keep usage under goal:

Global signal: If the substation detects that energy use is over goal, then for 10 minutes it will send a conserve signal to the houses under it. Every tick a house can decide whether or not to honor the request. If LowWaterMarks are being honored, then the house will always ignore a conserve request if its water heaters energy is too low.

Individual Signal: Every tick that a substation detects energy use is over goal, it sends a conserve signal to its houses. If a house chooses to respect the signal, then it gives a 10 minute wait to its water heater (which it can ignore if the LowWaterMark is reached).

Cooperative Mode: In this mode, household agents cooperate in order to meet the substations energy goal. Using a set of rules common to all the agents, they individually decide whether or not to claim an energy use token for ten seconds. While it maintains the token, that house is allowed to meet its energy needs.

In the cooperative strategy, the houses always follow the rules and respect assignment of the token which effectively means they always conserve. The Global and Individual strategies can also be run this way, but this leads to poor behavior (Has this been observed in practice too). If all houses respect the conserve signal, then they will eventually synchronize causing them all to turn on at once. The substation then sends out a conserve signal, since this pushes energy use over goal, and causes all water heaters to turn off. The simulation settles into a pattern of spikes where all heaters turn on, then periods of inactivity while all heaters conserve. For this reason, the model assigns each house a random chance to respect the conserve signal.

Details of Cooperative Mode The household agents in the cooperative mode operate using the *Belief-Desire-Intent* model of agent behavior:

Beliefs: States of the other agents (energy use, priority)
Desire: To meet their substations energy use goal while still maintaining quality of service to their residents
Intent: Run their appliances or to conserve

Every tick, households on the same substation send their estimated energy use and their priority. The priority is the energy level of their water heaters. This information forms each agents beliefs about the current needs of their neighbors. They then evaluate these beliefs against the desire to meet the energy goal of the substation while also keeping quality of service for their residents. To do this, each agent operates off the same set of rules shown in Figure 4.

The base load required from each house is subtracted from the energy goal. If any energy remains, then a list of houses is formed of those without a token. If any houses are below the LowWaterMark, then select the one that is furthest from its water heater's LowWaterMark. If that is the acting house, it claims the token. Next, remove the selected house from the list and subtract its energy needs. As long as energy remains, go through this process again until no house is below its LowWaterMark. If all houses are above LowWaterMark and energy remains, check if any house has a water heater below its Tank Energy Goal. Select the house that is closest to its goal. If it is the acting house, it claims the token. As with the previous process, remove the selected house from the list of remaining houses and subtract its energy needs. If any energy remains, repeat the steps of checking if a house is below its Tank Energy Goal. Acting by this rule set means that priority is given to those houses with the worst quality of service. This prevents "sacrificing" a house so that others may reach their goals faster. It also means that once a minimum is met, priority goes to the houses closest to reaching their goal in order to take them out of the equation. This simplifies the latter stages of planning.

Background

The Department of Energy's 2001 Residential Building Energy Consumption Survey (RECS) provides a consise breakdown

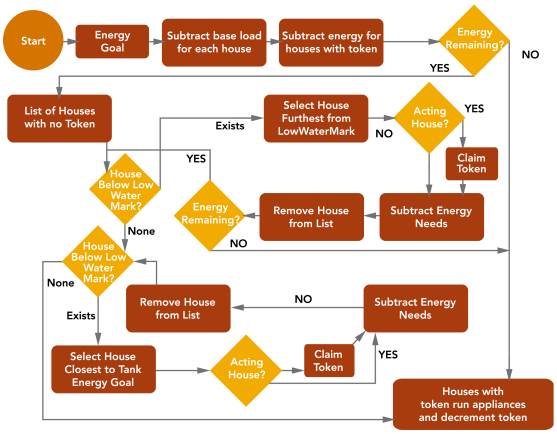

FIGURE 4. Workflow diagram of cooperative agent strategy

of consumer electric power consumption. Water Heating consumed an estimated 109 billion kWh out of the total 1,139.9 billion kWh demand by consumers, or approximately 9.1% [19]. Our hypothesis was that by changing the behavior of the water heater control system we could mitigate peak load conditions.

Based on nominal water heater characteristics from the the Department of Energy's 2005 Residential Building Energy Consumption Survey (RECS) a 40 Gallon tank with two 4.5 KW heating elements was selected for our simulation.

The first law of Thermodynamics, conservation of energy, provides the basis for calculating the water heater's load characteristics. Change in enthalpy is energy required to heat the water to the desired 50C output based on the following equation [20].

$$\Delta H = mc_p \Delta T \tag{1}$$

Where

ΔH is change in enthalpy, J
m is the mass of the water, kg
c_p is the spectific heat of water at constant pressure, J/mol*K
ΔT is the change in temperature, C

For the 160L tank with an incoming water temperature of 10C and outgoing temperature of 50C the change in energy is approximately 30.8 MJ. Using the tanks's 4.5 KW heating element will take approximately 114 minutes to heat the tank from an initial state.

The Air Conditioning, Heating, and Refrigeration Institute provides guidelines on nominal hot water usage based on activity [21]. The standard shower consumes 12 gallons of hot water, which in turn requires 9.6 MJ of energy to restore the tank to it's heated condition, or 35.6 minutes to return a fully heated state. Or in otherwords, if the shower lasted 5 minutes, during that time the tank experienced a loss of energy at the rate of approximately 32KW.

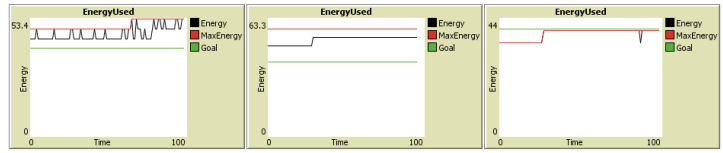

FIGURE 5. Comparison of Global, Individual, Coooperative Strategies

Simulation Setup

Although setup to run multiple substations at a time, they currently operate independently so experimental runs were done with a single substation. Runs were done with the substation having 5 houses. Energy goals of 30, 35, and 45 were used. Each house has a baseload of 5 kW/s and each water heater consumes 4.5 kW/s when heating water in the tank. Residents took showers randomly between 6am and 7am with a random duration from 5 to 15 minutes. However, the behavior of the residents were consistent for each strategy during a run. Worst case for every system is when there is only enough room in the energy goal to run one houses water heater.

DISCUSSION OF SIMULATION RESULTS

The cooperative strategy performed in a more stable way than the other two strategies by avoiding peaks in demand and periods of inactivity while waiting for conserve signals to expire. Figure 5 shows a representational snapshot of behavior for Global Signal, Individual Signal, and Cooperative stragegies the details of which are discussed below.

Global Signal: This strategy exhibits the behavior that the SmartGrid is trying to avoid: large initial spikes which trigger a conserve signal. After the 10 minute wait time, there is again a spike as all the water heaters turn on again then settle into chaotic behavior as water heaters decide to honor the signal or not.

Individual Signal: While it still exhibits an initial peak, the behavior is still much smoother than the global signal. Ticks over goal is still very high especially when water heaters fall below their LowWaterMark.

Cooperative: This strategy exhibits stable behavior and stays at the load goal or below until all water heaters have recharged themselves.

The model created for this research was not intended to be an exhaustive recreation of the power grid. Instead, creating a simpler model in a simple "instant feedback" framework allowed for easier implementation of conceptual models. The graphical nature of NetLogo also allows these strategies to be demonstrated to non-experts in an easier-to-understand manner. Also, even though it is simplified, the system exhibited realistic behavior observed in more complex simulations and in the real world.

Goal Energy	Strategy	Peak Energy	Ticks above goal	Recovery Time
40	global-signal	64.7	6633.8	11:39
40	individual-signal	58.4	4875.8	14:58
40	cooperative	39.5	0	14:34
45	global-signal	65.6	5058.4	10:18
45	individual-signal	59.3	3974.4	12:58
45	cooperative	44	0	10:33
60	global-signal	64.7	642.4	8:38
60	individual-signal	62	84.6	8:55
60	cooperative	57.5	0	8:29

TABLE 1. Average peak energy use, ticks over load goal, and recovery time for different strategies given a load goal.

CONCLUSION & FUTURE WORK

In this manuscript, we have shown that an intelligent sensor framework that supports agent-based sensing and control applications holds a unique promise in optimizing asset efficiencies and facilitate integration of distributed generation in the power grid. We have discussed three use cases as examples of what capabilities the agents provide, as well as presenting our initial design of the intelligent sensor framework. We have also demonstrated by means of simulations that the capabilities and coordination provided by use of agents in the power grid are real and significant. Now that our simulations have confirmed our initial hypothesis, in the future we will be working on implementing our software framework on real power grid hardware and making our software available to the power grid research community.

REFERENCES

[1] Lux Research, 2010. The Data Revolution: How Intelligent Hardware Will Drive the USD34 Billion Smart Grid. Tech. rep., December.

[2] Hammerstrom, D., Ambrosio, R., Carlon, T., DeSteese, J., Horst, G., and Kajfasz, R., 2007. Pacific Northwest GridWise Testbed Demonstration Projects. Tech. rep., Pacific Northwest National Laboratory, July.

[3] Stouffer, K., Falco, J., and Scarfone, K., 2008. *Guide to Industrial Control Systems (ICS) security. Technical Report Special Publication 800-82*. NIST.

[4] Smart Grid Interoperability Panel Cyber Security Working Group, 2010. *Guidelines for Smart Grid Cyber Security, NIST IR 7628*.

[5] D. Simon, C. C., 2005. "The squawk virtual machine: Java on the bare metal.". In Companion Proceedings ACM SIGPLAN, Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA), ACM Press.

[6] Python Software Foundation, 2011. Python v2.7.1 documentation, Feb. URL http://docs.python.org/.

[7] Lua. About lua. URL http://www.lua.org/about.html.

[8] Wiki, 2010. Lua versus python, Dec. URL http://lua-users.org/wiki/LuaVersusPython.

[9] Sun Inner Circle, 2008. Java technology takes a giant leap into the workd of microdevices, May. URL `http://www.sun.com/emrkt/innercircle/newsletter/0508/feature-tech.html`.

[10] Squeak, 2010. Squeak smalltalk, Jun. URL `http://www.squeak.org/`.

[11] Oracle Research Labs, 2006. New architectural forms to simplify construction of high performance or smallfootprint java virtual machines, Apr. URL `http://labs.oracle.com/projects/dashboard.php?id=12/`.

[12] Czajkowski, G., 2000. "Application isolation in the java virtual machine.". In Companion Proceedings ACM SIGPLAN, Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA), ACM Press.

[13] Smith, R. B., Cifuentes, C., and Simon, D. Enabling Java for Small Wireless Devices with Squawk and SpotWorld. Progress report, Sun Microsystems Labs, Menlo Park, CA.

[14] FIPA Organization. Fipa acl message structure specification. document id:sc00061. URL http://www.fipa.org/specs/fipa00061.

[15] Jini, 2010. Jini main page, Nov. URL `http://www.jini.org/wiki/`.

[16] BBN Technologies, 2009. Cougaar agent architecture, Aug. URL http://www.cougaar.org.

[17] Telecom Italia Spa, 2003. Jade - jade agent development framework, May. URL `http://jade.tilab.com/`.

[18] Wilensky, U., and Evanston, I., 1999. Netlogo: Center for connected learning and computer-based modeling. Tech. rep., Northwestern University, Evanston, IL.

[19] U.S. Energy Information Administration, 2001. End-Use Consumption of Electricity. Tech. rep., U.S. Energy Information Administration. URL `http://www.eia.doe.gov/emeu/recs/recs2001/enduse2001/enduse2001.html`.

[20] Octave Levenspiel, 1996. *Understanding Engineering Thermo*. Prentice Hall.

[21] U.S. Department of Energy, 2011. Sizing Storage and Heat Pump (with Tank) Water Heaters. Tech. rep., U.S. Department of Energy. URL `http://www.energysavers.gov/your\_home/water\_heating/index.cfm/mytopic=12990`.