Device Access Abstractions for Resilient Information Architecture Platform for Smart Grid

Abhishek Dubey, Gabor Karsai, Peter Volgyesi, Mary Metelko, Istvan Madari,
Hao Tu, Srdjan Lukic, and Yuhua Du

I. Introduction

Component-based software engineering (CBSE) has been accepted as a standard practice to develop robust, modular and maintainable software stacks for embedded systems [1]. The guiding principles of CBSE are interfaces with well defined execution models [2], compositional semantics [3] and model driven analysis [4]. In the past, our group has developed several such frameworks, including ARINC-653 component model [5], [6], which combines the principle of spatial and temporal partitioning with the interaction patterns derived from the CORBA Component Model (CCM) [7]. DREMS (Distributed Real-Time Embedded Managed Systems) component model [8] extended ACM to networked cyber-physical systems that can be used by several concurrent users, by allowing configurable real-time scheduling policies in addition to configurable secure information flow policies. A key theme across both ACM and DREMS work was a single threaded execution model for components, which helped avoid synchronization primitives that often lead to non-analyzable code and can cause run-time deadlocks and race conditions [9].

Recently, we have been studying the application of these methods for developing software for smart grid [10] using a platform called Resilient Information Architecture Platform for Smart Grid (RIAPS). Traditional power grids utilize a centralized model with large central power generation centers supplying power to customers [11] via transmission and distribution systems, where centralized stations handle the monitoring and control of predominately mechanical controls based on limited and time delayed data from remote devices. Power generally flows in only one direction. However, the smart grid of the future needs to fully manage the two-way flow of electricity and information [11]. One way to achieve this is to transform power grids into open application platforms where deployed physical devices, system infrastructure, and third party applications conform to industry standards and protocols, such as those being developed by IEEE and IEC [12].

An *open application platform* distributes the intelligence and control capability to local endpoints (or nodes) reducing total network traffic, improving speed of local actions by avoiding latency, and improving reliability by reducing dependencies on numerous devices and communication interfaces [12]. The platform must be multi-tasking and able to host multiple applications running simultaneously. Given such a system, the core functions of power grid control systems include grid state determination, low level control, fault intelligence and reconfiguration, outage intelligence, power quality measurement, remote asset monitoring, configuration management, power and energy management (including local distributed energy resources, such as wind, solar and energy storage) can be eventually distributed [13]. However, making this move requires extensive regression testing of systems to prove out new technologies, such as phasor measurement units (PMU) [11]. Additionally, as the complexity of the systems increase with the inclusion of new functionality (especially at the distribution and consumer levels), hidden coupling issues becomes a challenge with possible N-way interactions known and not known by device and application developers [13]. Therefore, it is very important to provide core abstractions that ensure uniform operational semantics across such interactions. In this paper, we describe the pattern for abstracting device interactions we have developed for the RIAPS platform. We provide evaluation of the use of this pattern in the context of a microgrid control application we have developed. However, first we provide a brief overview of RIAPS.

II. The Resilient Information Architecture Platform for Smart Grid

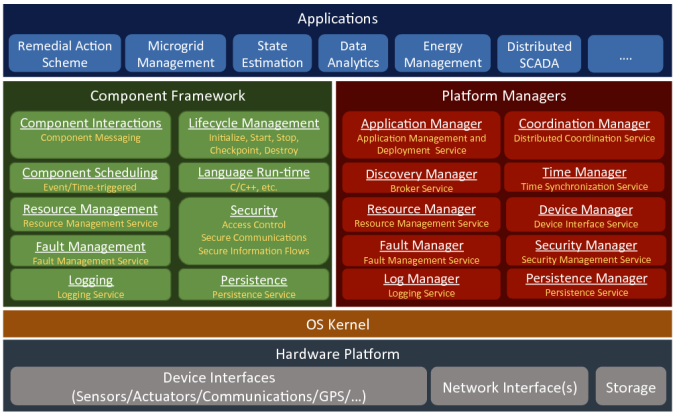

Fig. 1. RIAPS System Architecture Overview

RIAPS is an *open application platform* that distributes the intelligence and control capability to local endpoints reducing total network traffic, improving speed of local actions by avoiding latency, and improving reliability by reducing dependencies on numerous devices and communication interfaces. The platform is multi-tasking and able to host multiple applications running simultaneously. The key concept is to provide

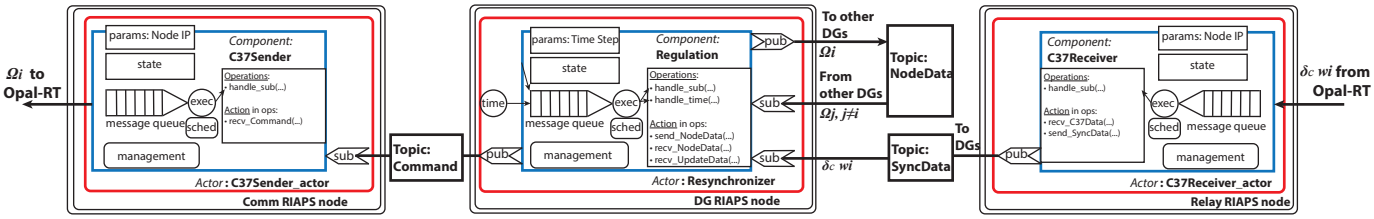

Fig. 2. Architecture of a RIAPS **actor** realizing phase/frequency regulation. The Actors receive sensor information about the microgrid simulated in a real-time simulator (Opal-RT). The control actions are then streamed back via custome device interfaces back to the Opal-RT. This hardware integrated simulation continues at 20Hz.

a "middleware" that enables each agent to communicate with others and focuses on specific grid issues, such as state estimation, remedial action schemes, and load shedding.

Figure 1 presents an overview of the middleware services provided by RIAPS to power systems application running on each remote node. The services include the Application Manager (that enables remote installation and management of the applications), the Distributed Coordination Manager (that implements fault-tolerant distributed service like leader election, consensus, coordinated actions, etc.), the Discovery Manager (which determines available connections among components on the same node or other operating nodes), the Time Manager (that provides high-precision timing and time synchronization services), the Resource Manager (monitors computing resources to ensure components and Platform Managers are able to run concurrently), the Fault Manager (that provides node-level fault management services), the Device Manager (that supports access to and management of attached input/output devices), the Security Manager (that handles authentication and manages keys and digital signatures), the Log Manager (that serves as a single entry point to all log activity on a node) and the Persistence Manager (that provides non-volatile data storage facility).

A. The RIAPS Component and Actor Model

A RIAPS component is a reusable unit of software that implements a set of operations for manipulating its state, and ports through which it communicates and interacts with other components. The operation of a component is analogous to a typical computer process in the sense that each component is limited to a single thread of computation. This thread is managed by a trigger method which is provided by the developer of the component. The trigger method monitors the state of the component and launches operations when 1) the state of the ports change, 2) a timer expires, or 3) an operation is completed. These operations implement the application logic of the component. The ports on the component are determined by the desired communication patterns which include asynchronous request/response, synchronous client/server, and publish/subscribe. The components on a particular compute node are managed by actors. An actor provides its components with the run-time code as well as the interfaces necessary to access platform services. Additionally the actor provides the capabilities to control and configure its components remotely.

One of the unique capabilities of RIAPS is the ability to interact with low-level devices like Modbus [14] and IEEE C37.118.2 synchrophasor data transfer protocol [15] as if they were part of yet another component, which allows other component developers to use message-driven interaction patterns uniformly. For example, consider the application shown in Figure 2. It implements a a microgrid synchronization algorithm described in [16]. The application has two actors. The first one is called C37receiver and it is deployed to the relay RIAPS node. C37receiver only has one component called C37device which abstracts the communication with an external device – the point of common coupling (PCC) relay using IEEE C37.118.2 synchrophasor data transfer protocol. The PCC relay constantly measures the voltage magnitude, frequency and phase difference between the main grid and microgrid. The measurement data along with the relay status are sent to the relay RIAPS node (an embedded computer) using C37 messages via the RIAPS device interface.

III. The RIAPS Device Interface Service

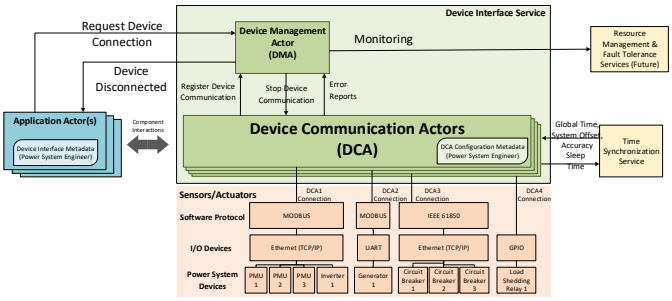

Fig. 3. Example of RIAPS Device Layers

RIAPS nodes vary in the type and number of hardware devices that are available for interactions based on the expected utilization of the node and the specific power system equipment connected to it. This configuration changes over time as systems are upgraded and re-purposed to meet the needs of the specific locations where they are installed. While devices might be available to a node, applications running at any specific time may only utilize a subset of the device capability available. There may also be more hardware capability available than the system resources can support simultaneously. Therefore, device interactions change over time and may be added and removed when needed.

Thus, RIAPS applications never interact with the power system devices directly, rather they interact via the Device Interface Service, which implements device-specific RIAPS

components (that are hosted in a Device Communication Actor, detailed below). The application components communicate with these component using the standard RIAPS interaction patterns: publish/subscribe and synchronous/asynchronous method invocation. In other words, the Device Interface Service encapsulates the power system devices into RIAPS components. The specific ports and interfaces implemented by these *Device I/O Components* (DIOC) are specific to the connected power system device, the lower-level protocol used, and the physical link used. This architecture ensures encapsulation of I/O devices so that application components can (a) access them using a *unified* interface and (b) the timing of interactions with the devices is highly *accurate*. To achieve these goals, the device components contains all the necessary drivers, resource arbitration methods, and a real-time scheduler for timed control actions. It is tightly integrated with the Time-synchronization Service [17] for executing device interactions on a globally synchronized timescale.

Example Device Configuration: Figure 3 shows an example of a possible RIAPS node configuration where there are different combinations of physical I/O connections and software protocols used to reach the power system devices. Each unique combination of a physical I/O connection and software protocol is a device connection point. The physical I/O hardware interface can support various industrial standards such as RS-232, RS-485, TCP/IP on Ethernet, I2C, or a simple GPIO. The low-level data communication can support various software protocols, such as Modbus, DNP3 and IEEE 61850. Multiple power devices can be physically connected to the same physical I/O hardware interface, for example an Ethernet connection running a Modbus software protocol could communication with a Phasor Measurement Unit (PMU), a Digital Fault Recorder (DFR) or inverters communicating using DNP3 on RS-485. Or a device connection could be as simple as a load shedding relay attached to a GPIO pin with no software protocol necessary. Since this configuration is unique to each node, a power system engineer will need to provide configuration information that identifies how the system is physically connected and how each power system device can be identified.

A. Architecture

The *Device Management Actor* (DMA) is in charge of servicing the device connection requests from the application actors, tracking the health of the power system devices, and stopping device communications when connected applications terminate. *Device Communication Actors* (DCA) are created for each device connection point available within the RIAPS node. Power system engineers will create Device Configuration Metadata (DCM) to describe how the specific node hardware is configured: what concrete power system devices are connected to this node, how they are configured, etc. Using this configuration metadata, each DCA will register itself with the DMA to inform it which power system devices are available through its device connection point. Upon registration, each DCA is initialized on demand.

Application actors are deployed together with Device Interface Metadata (DIM) that is created by a system manager (a power system engineer) to identify the specific power system devices that the application will be utilizing. A RIAPS application is designed and developed with the assumption of specific *types* of power system devices, but when it is deployed it has to connect to specific *instances* of those devices. This binding between the abstract and the concrete device(s) is represented in the DIM and it is part of the configuration files deployed with the application. Note that the implementation of the DIOC-s are not part of the application - they reside in the DCA-s, and the application components are bound to these DIOC-s at run-time.

Application actors send the DMA a connection request to link with the power system devices. If a DCA with the requested power system device is available (registered), the DMA provides connection information to the application actor so that it can start communicating with the appropriate DCA directly. The setting up of the connection between the application actor's components the the DIOC-s of the DCA(s) are facilitated through the RIAPS Broker Service (just like connections between any other RIAPS components) [10]. Each DCA reports when hardware errors occur to allow the DMA to monitor the health of the device connections. If a device becomes unavailable due to hardware failures, the DMA informs the attached application actor that the device disconnected. If all application actors connected with a power system device are removed from the node, the DMA will tell the DCA to stop all device communications for that specific device.

Once the connection has been established between an application actor and the appropriate DCA, direct communication will occur between application actors and DCA to manage the real-time data access. Specific device type API-s will be defined by power system device classes. These device type classes will identify the actions and configuration available for the specified device. These device specific API-s will be based on the power system device type and capability level, and will be determined in the future as equipment capability is added to the RIAPS platform. For instance, application actors can listen for desired sensor data or command actuation of a connected device. Some actuation commands can include a request to provide a notification back to the application actor after future scheduled actuation has completed. These notifications could be a published message by the appropriate DCA.

The DCA translates to and from the appropriate physical and software protocol combination for the application actors and, if needed, add timestamps to the power device data based on the global time provided by the time-synchronization service. For periodic actions or scheduled outputs, each DCA manages the real-time scheduling needs for the devices associated with their specific device connection point by requesting global time information and scheduling sleep times when needed. Overall, this architecture can support the following interaction patterns between the physical power system device and application actors.

- **Sporadic input:** When the sensory I/O device generates a new sample, it is time-stamped (possibly by the device itself or with global time from the Time Synchronization Service) and this sample is sent to interested application(s) through published messages. Another option is that the data is stored in a queue of samples or as a single sample (with 'most-recent-overrides' semantics), and the application will query this from the service.
- **Periodic input:** When a periodic sensor is capable of producing a stream of data, then the device interface service is commanded to 'turn on the data pump' (meaning that the continuous streaming is enabled for that sensor). It is assumed that the sensor pushes samples into the Device Interface Service that timestamps and then distributes it to interested applications.
- **Sporadic output:** An application calls the service to generate an output for an actuator. The transfer takes places as fast as possible. The caller application may or may not wait for the transfer to complete. In the latter case, the application actor may have registered a request to receive notification about the success of the transfer, asynchronously.
- **Periodic output:** An application commands the service to set up a periodic actuator output activity and provides an initial value to send. As a result, the service launches a looping thread that reads the value to be sent from a sample location, sends that value to the actuator, then repeats this with a specific frequency. The application can asynchronously update the sample location with a new value - the updated value will be used starting from the next transfer.
- **Scheduled output:** An application needs to execute actuation commands at a specific point (or at periodically scheduled specific points) of (physical) time. The application commands the service to schedule the actuation action and provides the data to be sent. The service schedules this request and when the physical time arrives, the request is executed. If the request is repeating, it is rescheduled for the next run.

IV. Case study - the C37 Device Wrapper and the Microgrid Synchronization Application

The RIAPS framework supports simplified bi-directional C37.118.1 communication interfaces for integrating measurement and control applications with industrial phasor measurement units (PDU) and/or phasor data concentrators. This capability is provided by two device components in the framework, acting as intelligent bridges and multiplexors between the RIAPS pub/sub communication infrastructure and the external C37 connections.

The C37Sender device component is used when the RIAPS application acts as a PDU (see fig 2) and wants to provide streaming synchrophasor frequency, rate of change of frequency (ROCOF) and other status information for external sinks. The component offers three subscribe ports for other RIAPS components for (1) updating the PDU configuration state, (2) updating the header/identification information and (3) receiving measurement data. Only the last port is used for continuous streaming. Due to current implementation-level limitations, the data on these ports need to be encoded properly as defined by the IEEE Std C37.118.1-2011 standard.

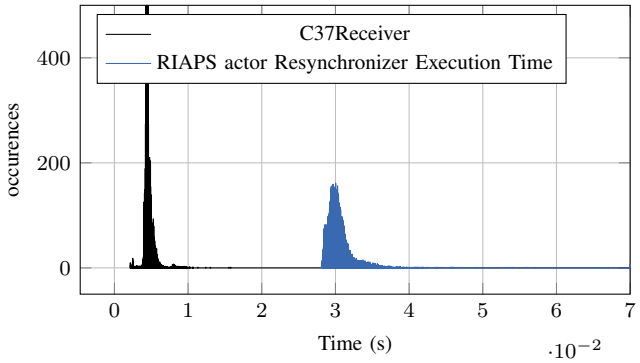

Fig. 4. Time for RIAPS actor C37Receiver to receive and process the C37 message. The experiment contains 20000 data points. The mean value for transmission and processing timeof C37 message is 4.5 ms and the std=1.27 ms. The Mean value of total execution of resynchronization component is 30.27 ms (run at 20Hz).

Internally, the device component uses a multithreaded server pattern: one dedicated thread is listening on the PDU port - in C37 the PDC initiates the TCP connection to the PDU -, and each active PDU is handled by a dedicated, dynamically spawned thread. Incoming RIAPS data is always broadcasted to all connected PDCs using a simple message queue per PDC thread. The RIAPS event handler, which cannot use blocking primitives, append the data messages to these queues with some capacity threshold, above which it drops the message arriving from the RIAPS application. The C37Receiver component is simpler, it can connect to one external PDU, only, thus it relies on a single device thread for executing a blocking read on the TCP socket. It provides three publish RIAPS ports for other components with the same but reversed logic as described above.

Fig 4 shows the timing of the C37 actor as measured during an experiment with the resynchronizer application shown in figure 2. The whole application runs at 20 Hz, receiving data from a real-time simulation of a micro-grid that we have buit. Details of the application logic and the simulator are available in [16]. During each run, the data is received via C37 from the real-time simulator, processed through other RIAPS actors and then eventually sent back to the microgrid simulator for the actuation. The stability of this application has already been shown in prior work [16]. Here we show the timing of the C37 device interface (fig 4) and the over all timing of the resynchronization component (fig 2).

V. Conclusion

Component frameworks like RIAPS are going to increasingly become mainline as we witness the large scale integration of cyber and physical aspects of our infrastructure, especially when it is distributed across large geographical areas.

These frameworks provide a "smart operating system", which allows us to go beyond the traditional SCADA frameworks and disperse the "intelligence" throughout the infrastructure. This paper briefly described the RIAPS framework and the mechanisms required to ensure that the "app" developers can correctly interact with the physical devices such as Phasor Measurement Units. For this purpose, the device interface provides publish-subscribe interfaces to the other components, while internally encapsulates the logic required to realize the sporadic input, periodic input, sporadic output, periodic output, or scheduled output interaction patterns. We presented a case study with respect to a C37 device used in a microgrid resynchonizer application. Our ongoing work is continuing to integrate other device interfaces, for example Modbus. We are also working on developing a C and `capnproto` [18] based device framework implementation that can achieve frequencies higher than what was shown in this paper (20 Hz).

REFERENCES

[1] G. T. Heineman and W. T. Councill, "Component-based software engineering," *Putting the pieces together, addison-westley*, p. 5, 2001.

[2] A. Basu, B. Bensalem, M. Bozga, J. Combaz, M. Jaber, T.-H. Nguyen, and J. Sifakis, "Rigorous component-based system design using the bip framework," *IEEE software*, vol. 28, no. 3, pp. 41–48, 2011.

[3] H. Schmidt, "Trustworthy components—compositionality and prediction," *Journal of Systems and Software*, vol. 65, no. 3, pp. 215–225, 2003.

[4] P. S. Kumar, A. Dubey, and G. Karsai, "Colored petri net-based modeling and formal analysis of component-based applications." in *MoDeVVa@ MoDELS*. Citeseer, 2014, pp. 79–88.

[5] A. Dubey, G. Karsai, and N. Mahadevan, "Formalization of a component model for real-time systems," 04/2012 2012.

[6] ——, "A Component Model for Hard Real-time Systems: CCM wwith ARINC-653," *Software: Practice and Experience*, vol. 41, no. 12, pp. 1517–1550, 2011. [Online]. Available: http://www.isis.vanderbilt.edu/sites/default/files/Journal_0.pdf

[7] N. Wang, D. C. Schmidt, and C. O'Ryan, "Overview of the corba component model," in *Component-Based Software Engineering*. Addison-Wesley Longman Publishing Co., Inc., 2001, pp. 557–571.

[8] D. Balasubramanian, A. Dubey, W. Otte, T. Levendovszky, A. Gokhale, P. Kumar, W. Emfinger, and G. Karsai, "DREMS ML: A Wide Spectrum Architecture Design Language for Distributed Computing Platform," *Sci. Comput. Program.*, vol. 106, no. C, pp. 3–29, Aug. 2015. [Online]. Available: http://dx.doi.org/10.1016/j.scico.2015.04.002

[9] W. Otte, A. Dubey, S. Pradhan, P. Patil, A. Gokhale, G. Karsai, and J. Willemsen, "F6com: A component model for resource-constrained and dynamic space-based computing environments," in *Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC), 2013 IEEE 16th International Symposium on*, June 2013, pp. 1–8.

[10] S. Eisele, I. Mardari, A. Dubey, and G. Karsai, "Riaps: Resilient information architecture platform for decentralized smart systems," in *2017 IEEE 20th International Symposium on Real-Time Distributed Computing (ISORC)*, May 2017, pp. 125–132.

[11] E. P. R. Institute, "Needed: A grid operating system to facilitate grid transformation," PDF, June 2011. [Online]. Available: https://www.smartgrid.gov/files/Needed_Grid_Operating_System_to_Facilitate_Grid_Transformati_201108.pdf

[12] EPRI, "Transforming smart grid devices into open application platforms," Electric Power Research Institute Report 3002002859, July 2014. [Online]. Available: http://www.epri.com/abstracts/Pages/ProductAbstract.aspx?productid=000000003002002859

[13] J. D. Taft and M. G. Seewald, "Developing a distributed intelligence architecture for smart grids," in *21st International Conference on Electricity Distribution*, no. 1284. Frankfurt: CIRED, June 2011, p. 4.

[14] I. Modbus, "Modbus application protocol specification v1. 1a," *North Grafton, Massachusetts (www. modbus. org/specs. php)*, 2004.

[15] K. Martin, D. Hamai, M. Adamiak, S. Anderson, M. Begovic, G. Benmouyal, G. Brunello, J. Burger, J. Cai, B. Dickerson *et al.*, "Exploring the ieee standard c37. 118–2005 synchrophasors for power systems," *IEEE transactions on power delivery*, vol. 23, no. 4, pp. 1805–1811, 2008.

[16] Y. Du, H. Tu, S. Lukic, D. Lubkeman, A. Dubey, and G. Karsai, "Implementation of a distributed microgrid controller on the resilient information architecture platform for smart systems (riaps)," in *2017 North American Power Symposium (NAPS)*, IEEE. Morgantown, WV: IEEE, 09/2017 2017.

[17] P. Volgyesi, A. Dubey, T. Krentz, I. Madari, M. Metelko, and G. Karsai, "Time synchronization services for low-cost fog computing applications," *Rapid Systems Prototyping (RSP). IEEE, IEEE*, 2017.

[18] K. Varda, "Cap'n proto: Introduction," *Sandstorm,[Online]. Available: https://capnproto. org/index. html.[Använd 31 mars 2017]*, 2013.