Towards a Resilient Information Architecture Platform for the Smart Grid: RIAPS

Gabor Karsai, Vanderbilt University (PI)
In collaboration with
Abhishek Dubey (Vanderbilt)
Srdjan Lukic (NCSU)
Anurag Srivastava (WSU)

Supported by DOE ARPA-E under award DE-AR0000666

https://riaps.isis.vanderbilt.edu/
The Energy Revolution: Big Picture

From centralized to a decentralized and distributed energy systems

- Changing Generation Mix
- Transactive Energy
- Electrical Cars
- Decentralization
The control picture has not changed

Communication Network

Centralized SCADA supported by a utility company
The control picture has not changed

Communication Network

Problems
- Integration challenge
- Reliability issues
- Management issues
RIAPS Vision

Communication Network

Example Power System: IEEE 30 bus system

Control Room

RIAPS Node:

RIAPS Computing Platform
Sensors
Actuators

Network
I/F
RIAPS Vision

- Push computation to the edge
- Enable common technology stack across the ecosystem
- Provide core services to enable the rapid development of smart apps
RIAPS Software Platform

• At the core of the RIAPS vision is a reusable technology stack to run Smart Grid applications. A software platform defines:
  – Programming model (for distributed real-time software) on embedded nodes dispersed throughout the power grid
  – Services (for application management, fault tolerance, security, time synchronization, coordination, etc.)
  – Development toolkit (for building and deploying apps)
# A Reusable Software Platform for Smart Grid

<table>
<thead>
<tr>
<th>Applications</th>
<th>Component Framework</th>
<th>Platform Managers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Remedial Action Scheme</td>
<td><strong>Component Interactions</strong> Component messaging</td>
<td><strong>Application Manager</strong> Deployment Service Application Management Service</td>
</tr>
<tr>
<td>Microgrid Management</td>
<td><strong>Component Scheduling</strong> FIFO, Priority, EDF</td>
<td><strong>Coordination Manager</strong> Distributed Coordination Service</td>
</tr>
<tr>
<td>State Estimation</td>
<td><strong>Lifecycle Management</strong> Initialize, Start, Stop, Checkpoint, Destroy</td>
<td><strong>Discovery Manager</strong> Broker Service</td>
</tr>
<tr>
<td>Data Analytics</td>
<td><strong>Language Run-time</strong> C/C++, etc.</td>
<td><strong>Time Manager</strong> Time Synchronization Service</td>
</tr>
<tr>
<td>Distributed SCADA</td>
<td><strong>Fault Management</strong> Fault Management Service</td>
<td><strong>Device Manager</strong> Device Interface Service</td>
</tr>
<tr>
<td></td>
<td><strong>Logging</strong> Logging Service</td>
<td><strong>Fault Manager</strong> Fault Management Service</td>
</tr>
<tr>
<td></td>
<td><strong>Persistence</strong> Persistence Service</td>
<td><strong>Security Manager</strong> Security Management Service</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Log Manager</strong> Logging Service</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Persistence Manager</strong> Persistence Service</td>
</tr>
</tbody>
</table>

## OS Kernel

## Hardware Platform

- **Device Interfaces** Sensors/Actuators/Communications/GPS/...
- **Network Interface(s)**
- **Storage**
Core RIAPS Concepts: RIAPS System

- RIAPS Nodes: Networked Embedded Computers
- Applications consist of actors that contain components
- Component communicate and interact using well-defined patterns: publish/subscribe + client/server
- Expected scale: $\sim10^2$ nodes
Design: Component Framework

- **Software components** are the reusable building blocks of applications (actors group them into a single process)
- Components have a *state* and interact via *ports*
  - *Receiver* of messages (‘subscriber’)
  - *Server* of client requests
  - *Sender* of messages (‘publisher’)
  - *Client* of servers
- Components are single-threaded: one operation at a time
- Components are triggered by the arrival of a *message*, a *request*, or a *timer event*

- Component triggering logic is encapsulated in a function that can include complex decisions

**Benefits:** Reusable components + concurrency is handled in the framework (not in the ‘business logic’) + lends itself to timing analysis
Design: Component Framework

Example: 3 actors, with 2+1+1 components, interacting via pub/sub and client/server patterns

Implementation languages:
Python, C++
Design: Platform Services

• Application Deployment and Management
  – Function: Remotely installs and manages apps

Benefit: Authoritative control over all software deployed on the RIAPS network.
Design: Platform Services

• Discovery: The Broker Service
  – The ‘matchmaker/housekeeper’ – how the components/actors of an app find each other on the network

**Benefit:** Actors of a RIAPS app can come and go at any time – they are still able to connect to the group reliably.
Design: Platform Services

• Discovery: The Broker
  – A fault-tolerant distributed database where component register themselves and look up other components
  – Publishers ↔ Subscribers + Clients ↔ Servers
  – Implementation: distributed hash table
Design: Platform Services

• Time Synchronization
  – Maintains a cluster-wide synchronized notion of time
  – Applications can: (1) query the global time, (2) sleep until a specified point in time, (3) query the status of the service
  – Architecture:
    • Use PTP (IEEE-1588)
    • Some nodes may have a GPS
    • GPS clock is distributed
    • Fallback: NTP
    • Accuracy: ~10 usec

Benefit: Precisely synchronized time base available to all apps on the RIAPS network.
Design: Platform Services

• Device interface
  – Encapsulates ‘power system devices’ that use specific protocols (e.g. Modbus, DNP3, IEC 61850 etc.) and hardware interfaces (RS-232, TCP/IP, etc.) and provides a (RIAPS-compliant) messaging interface to the device

  ▶ Device interactions:
    ▶ Sporadic input: sensor reading at an arbitrary time
    ▶ Periodic input: periodic sensor reading (stream)
    ▶ Sporadic output: actuator command at an arbitrary time
    ▶ Periodic output: actuator periodically updated
    ▶ Scheduled output: actuator is updated at a specific point in time

Benefit: Portable applications – device dependencies are encapsulated in the service.
Design: Platform Services

• Distributed coordination
  – For coordinating applications distributed on the network
  – Features:
    • *Group membership*: join/leave group, query membership, get notified when membership changes
    • *Leader election*: elect a leader for centralized functions, when leader becomes unavailable elect another one automatically
    • *Distributed consensus*: participants agree on a ‘value’
    • *Time-coordinated action*: execute a control action on many nodes simultaneously (up to time synchronization accuracy)
  – Algorithms: Paxos/RAFT

*Benefit: Reusable implementation of difficult algorithms – available as a service.*
Design: Platform Services

- Resource management
  - Keeps track of resource usage (CPU, memory, files space, I/O)
  - Manages quotas and access
  - Signal errors/terminates applications if resource restrictions are violated

- Logging
  - Efficient, low-overhead logging of events in apps and managers
  - Global management of all logs

- Persistence
  - Efficient, low-overhead database for node-local real-time data
  - Global management of the database

- Fault management
  - Monitors apps/managers/system for faults
  - Mitigates fault effects (e.g. automatic restart, checkpoint, etc.)

- Security management
  - Secure information flows among app components
  - Global management of security keys

Benefit: Complex housekeeping functions – apps don’t need to implement them.
Design: Model-driven Development

• Goal: Average software developers are productive in developing complex RIAPS apps
• Developers build:
  – Application ‘business logic’ – the algorithms
  – Models to represent the components and their composition to form an app
• Toolchain generates:
  – Intermediate code, software engineering artifacts
• Model-based toolchains are effective (Simulink/Stateflow)
  – But they are ‘closed’ and not suited for distributed systems

**Benefits:** Developer can focus on the core logic of the application (the ‘algorithms’) – the composition and configuration is done on a higher-level of abstraction.

Institute for Software Integrated Systems
World-class, interdisciplinary research with global impact.
VANDERBILT UNIVERSITY
Design: Model-driven Development

- Approach:
  - Use a simple, text-based language for the models (diagrams, if needed, can be rendered automatically)
  - Integrated it with the code-based IDE (Eclipse) where the application logic is entered (as C++ code)
  - Develop code generators and integrate them into the IDE (for a seamless workflow)
  - Prototype: Eclipse
Application 1: Response Based Remedial Action Scheme (WSU)

- RAS is a key mechanism to protect electric power grid, generally used as the last line of automatic defense.
- Existing RAS are pre-determined, inflexible and do not factor in changing system conditions and might take control actions good for small system but not optimal for the overall power grid.
- RIAPS will enable dynamic coordinated response based RAS (DCRB-RAS), which will use measurements, changing network conditions, control settings to dynamically decide control decisions.

![Diagram of RAS system](image)
Application 1: Response Based Remedial Action Scheme (WSU)

Two applications:

RAS I for managing wind generation: curtailment
- Data acquisition actor: Protocol conversion, periodic and event data input, time stamping, buffer input data, time aligning
- DLSE actor: Noise filtering, bad data, topology processing, WLS
- RAS actor: Initialization, obtain state variable, optimization, solution update, generate control actions

RAS II for under-frequency control: load shedding
Application 2:
Microgrid Islanding (NCSU)

- Application of interest: Formation and interactions of microgrids on a distribution feeder
- Focus: power management
- Main application scenario:
  - Unplanned transition from grid-connected to islanded mode and re-synchronization.
  - Distributed control and protection framework will be used to implement a fast transition scheme
Demo of an Early Prototype: Synchronization Application

https://riaps.isis.vanderbilt.edu/blog/
Expected outcomes

- The platform will enable developers – sanctioned by utilities - to build reusable components and applications
- The platform specification and its prototype implementation is open source, but industrial partners will provide software development services for it
- A new open standard that will change how software for the smart grid is developed