Exceptional service in the national interest

Sandia National Laboratories

SAND2016-11629PE

IDC Reengineering Phase 2

Elaboration Phase Prototyping Review

Ryan Prescott

14 November 2016

U.S. DEPARTMENT OF ENERGY

National Nuclear Security Administration

Sandia National Laboratories is a multi-mission laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Outline

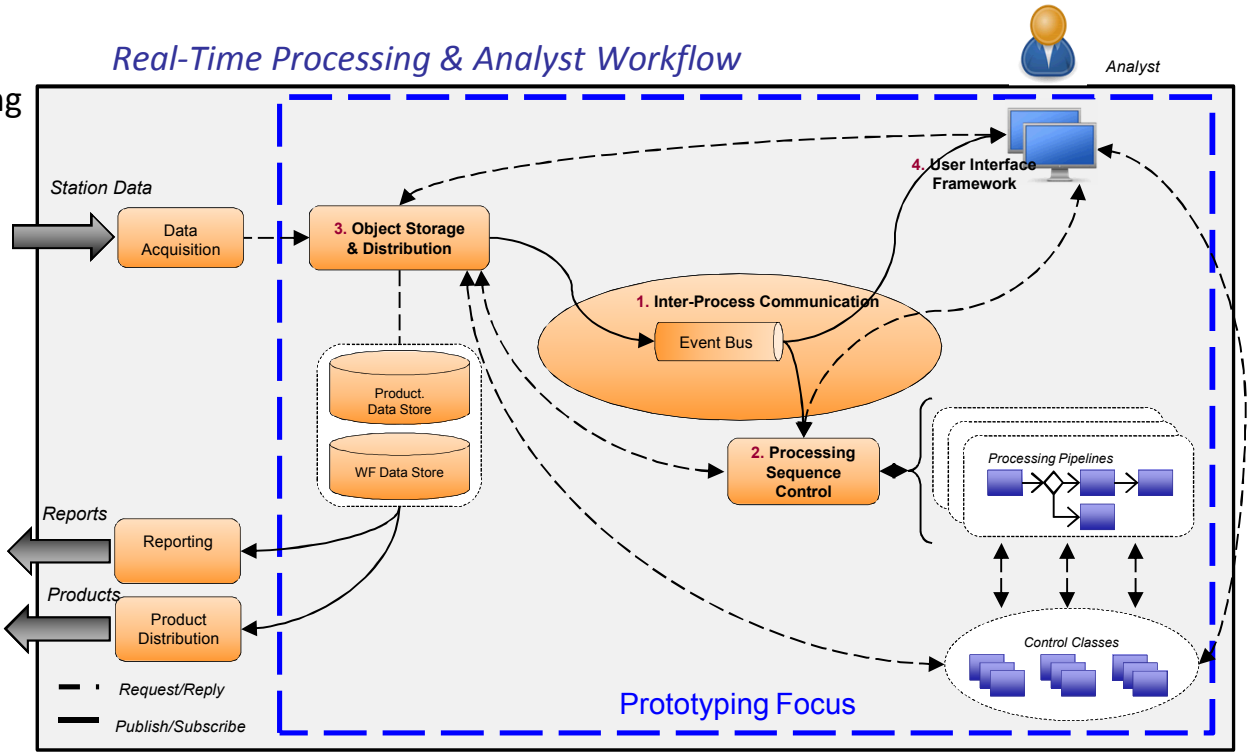
- Prototyping Overview
- Prototyping Focus
- Software Mechanism Prototypes
  - Technology Summary
  1. Inter-Process Communication
  2. Processing Sequence Control
  3. Object Storage & Distribution
  4. User Interface Framework
- Key Features Prototypes
  - Provenance – Event Sourcing
- Summary

Prototyping Overview

- The Elaboration Phase Statement of Work includes Executable Architecture prototyping as a deliverable
- The intent of this work is to support definition of the System architecture
  - Early design consideration of non-functional requirements
    - E.g. scalability, extensibility
  - Key technology decisions
    - Software mechanisms/frameworks
    - Development environment & tooling
  - Early elaboration of key feature designs
    - E.g. provenance

Prototyping Focus

- The Elaboration Phase project team developed a number of prototypes focusing on key software mechanisms and features defined in the architecture
- Software Mechanisms
  1. Inter-Process Communication
  2. Orchestration – Processing Sequence Control (PSC)
  3. Data Persistence – Object Storage & Distribution (OSD)
  4. User Interface Framework
- Features
  - Provenance – Event Sourcing



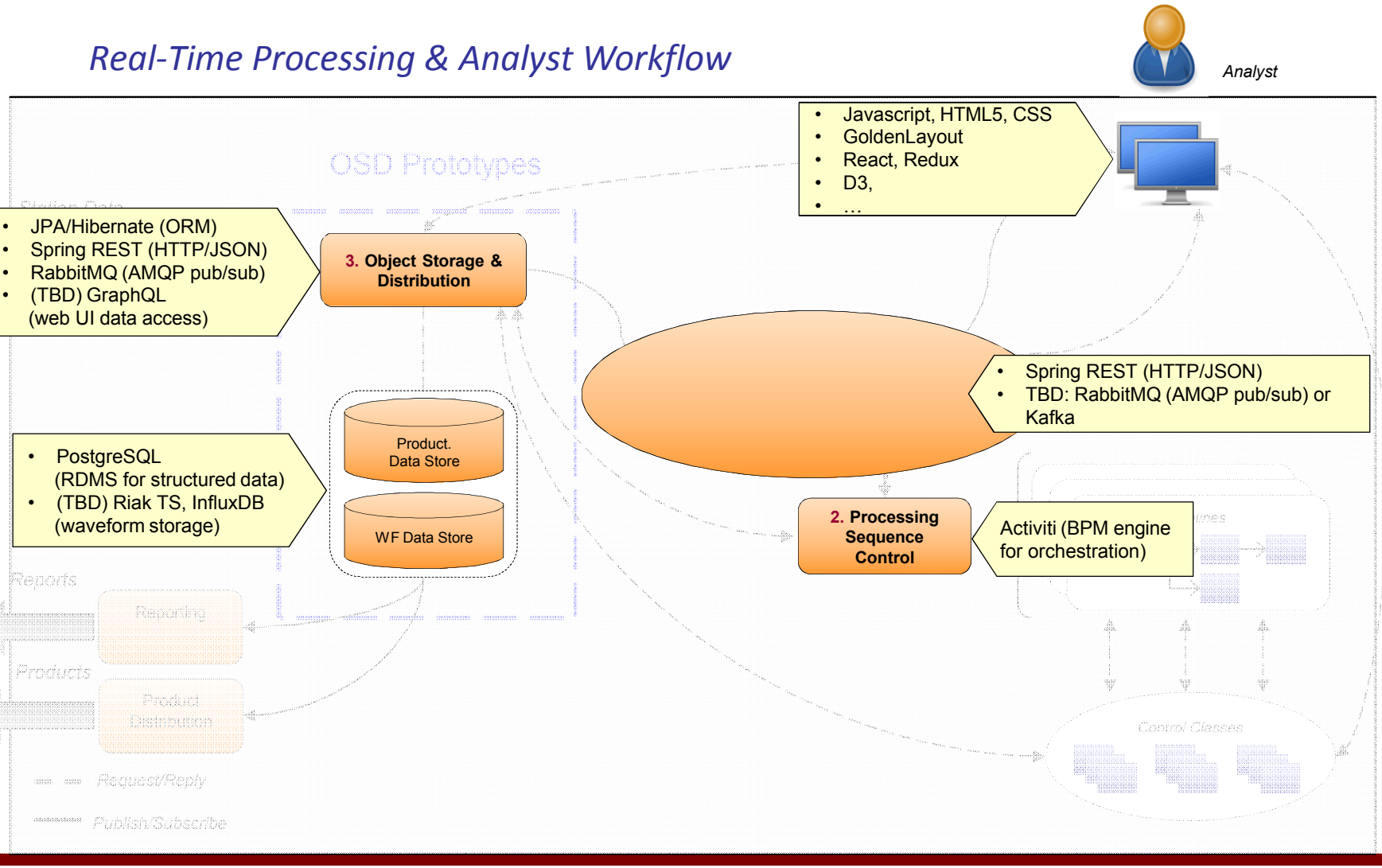
- Prototyping Overview
- Hybrid Architecture Context
- Prototyping Focus
- **Software Mechanism Prototypes**
  - Technology Summary
  1. Inter-Process Communication
  2. Processing Sequence Control
  3. Object Storage & Distribution
  4. User Interface Framework
- Key Feature Prototypes
  - Provenance – Event Sourcing
- Summary

SOFTWARE MECHANISM PROTOTYPES

Software Mechanism Prototypes – Technology Summary

Sandia National Laboratories

Real-Time Processing & Analyst Workflow



Inter-Process Communication – Overview

Sandia National Laboratories

- The Inter-Process Communication mechanism provides the protocols, interfaces and data encoding necessary to communicate data and control information between components deployed as part of the distributed architecture
- Design Drivers
  - Support the scalability and performance requirements defined in the SRD & SSD
    - Support distributed processing model for horizontal scalability & loose coupling
    - Message encoding/transmission latency and throughput aligned with anticipated data processing volume and reporting timelines
  - Support the communication patterns defined in the architecture
  - Enable communication between software components implemented in any of the languages/technologies defined for the system
    - Java, C, C++, Python, javascript

Inter-Process Communication – Technology Selection

Sandia National Laboratories

Messaging

- Evaluated a broad range of technologies including e.g.
  - Enterprise Service Bus (MuleESB, Camel)
  - AMQP/STOMP Brokered Solutions (ActiveMQ/Apollo, Qpid, RabbitMQ)
  - Others: DDS (OpenSplice, RTI), Kafka
  - HTTP
- Selected HTTP for request/reply communication
  - **Rationale**: ubiquitous standard protocol for request/reply communication minimizing effort to integrate components across technologies, languages, etc.
  - Well suited for OSD data access and control class/plugin APIs
- Prototyped RabbitMQ (AMQP) for publish/subscribe communication
  - Widely-used open standard (AMQP) offering a good trade-off between features (message queuing & routing, delivery guarantees) and complexity
  - Currently evaluating Apache Kafka as an alternative based on performance concerns with durable queues at scale

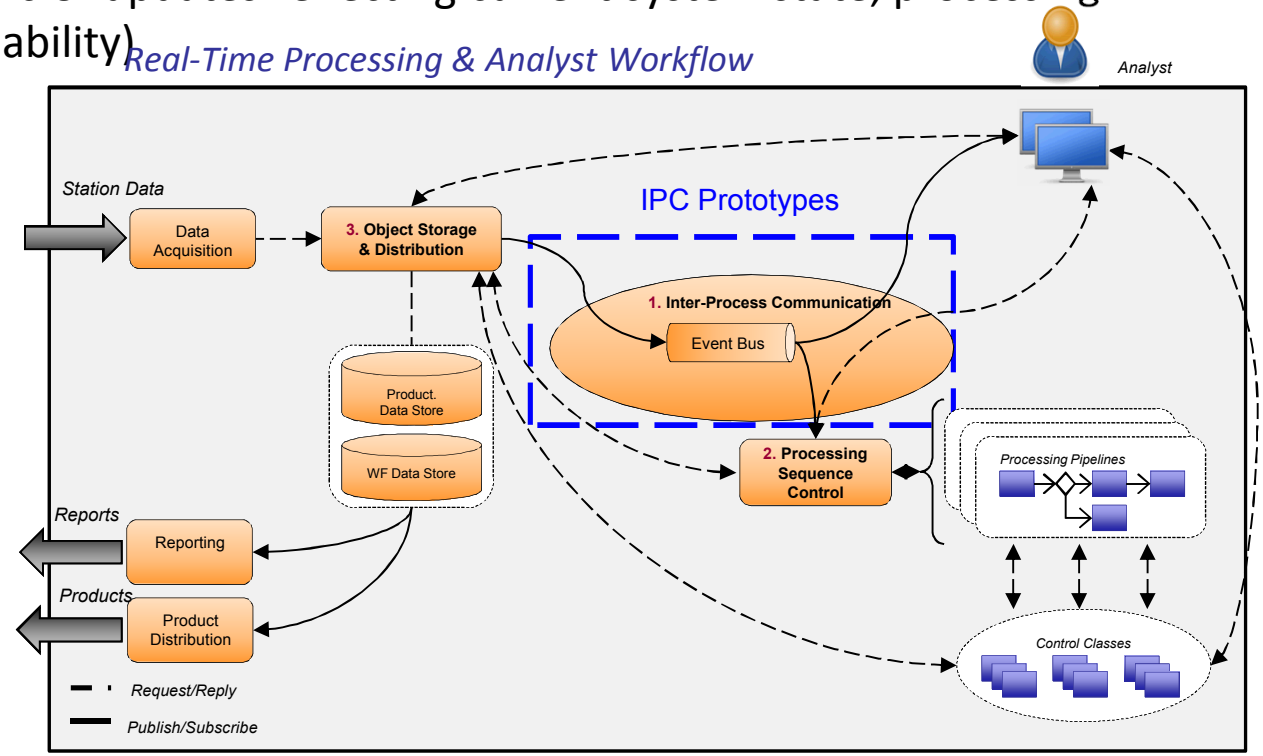
Data Encoding

- Evaluated a broad range of message formats including Apache Thrift, Apache Avro, Google protocol buffers, JSON
- Selected JSON for structured data
- Rationale:
  - Ubiquitous, human-readable, open standard with relatively compact syntax (relative to e.g. XML)
  - Binary formats (Thrift, protobuf) impose significant limitations on data model design (e.g. inheritance)
- Message format for waveform data is TBD
  - Prototypes used JSON and protocol buffers
  - Performance evaluation and exploration of compression/alternate format options is ongoing

Inter-Process Communications – Messaging Patterns

- The architecture defines two primary IPC patterns:
  - **Request/Reply** is used to access control classes and other processing components (e.g. plugins, OSD) directly via network service requests
  - **Publish/Subscribe** is used for event-driven processing, where interested components subscribe for notifications of specific processing events in the System (e.g. dynamic UI updates reflecting current system state, processing based on data availability)

Real-Time Processing & Analyst Workflow



Inter-Process Communication – Elaboration Prototypes

Sandia National Laboratories

- Developed HTTP/JSON APIs as part of the OSD mechanism, providing access to select stored data entities (e.g. waveform, channel, station, signal detection)
- Developed HTTP/JSON APIs for the Signal Detection control class, STA/LTA detector & FK plugins as part of a PSC prototype
- Completed initial performance benchmarks of RabbitMQ/JSON publish/subscribe messaging
- Implemented a RabbitMQ/JSON event bus as part of the Event Sourcing prototype (see Event Sourcing section)
- Note: Implemented data update subscriptions within the web User Interface prototype using websockets pending further evaluation of web UI communication options (see Follow-On Work slide below)
- Kafka prototype underway

Inter-Process Communication – Follow-On Prototyping Work

Sandia National Laboratories

- Select data format for communication of waveform data
  - Evaluate compression
  - Evaluate binary formats
- Web UI data communication
  - Select technologies for data communication with the web-based user interface, providing:
    - Efficient data access APIs between browser and backend
    - Data subscriptions for dynamic display updates
  - Candidates: GraphQL, Falcor, Websockets

Object Storage & Distribution – Overview

Sandia National Laboratories

- The Object Storage and Distribution (OSD) mechanism provides interfaces for access to persistent data in the database (CRUD), as well as distribution of shared data across processing components
- The OSD Includes two primary capabilities:
  1. Common Object Interface (COI) implementation, supporting access to stored data across distributed processing components, languages and technology stacks, independent of the underlying storage solution (relational, document, key-value, etc.)
  2. Subscription-based notification of data events (creation, modification, deletion) within the System (i.e. publish subscribe)
- Design Drivers
  - Provide scalable storage with performant Create/READ/Update/Delete (CRUD) access, supporting both structured and unstructured data (e.g. raw waveform)
  - Provide access to stored data via the COI data model abstraction with support for the languages defined for the system (C, C++, Java, Python, javascript)
  - Insulate components of the system from the technology and schema of the underlying data storage solution so that these can be modified with minimal impact to the System

Object Storage & Distribution – Technology Selection

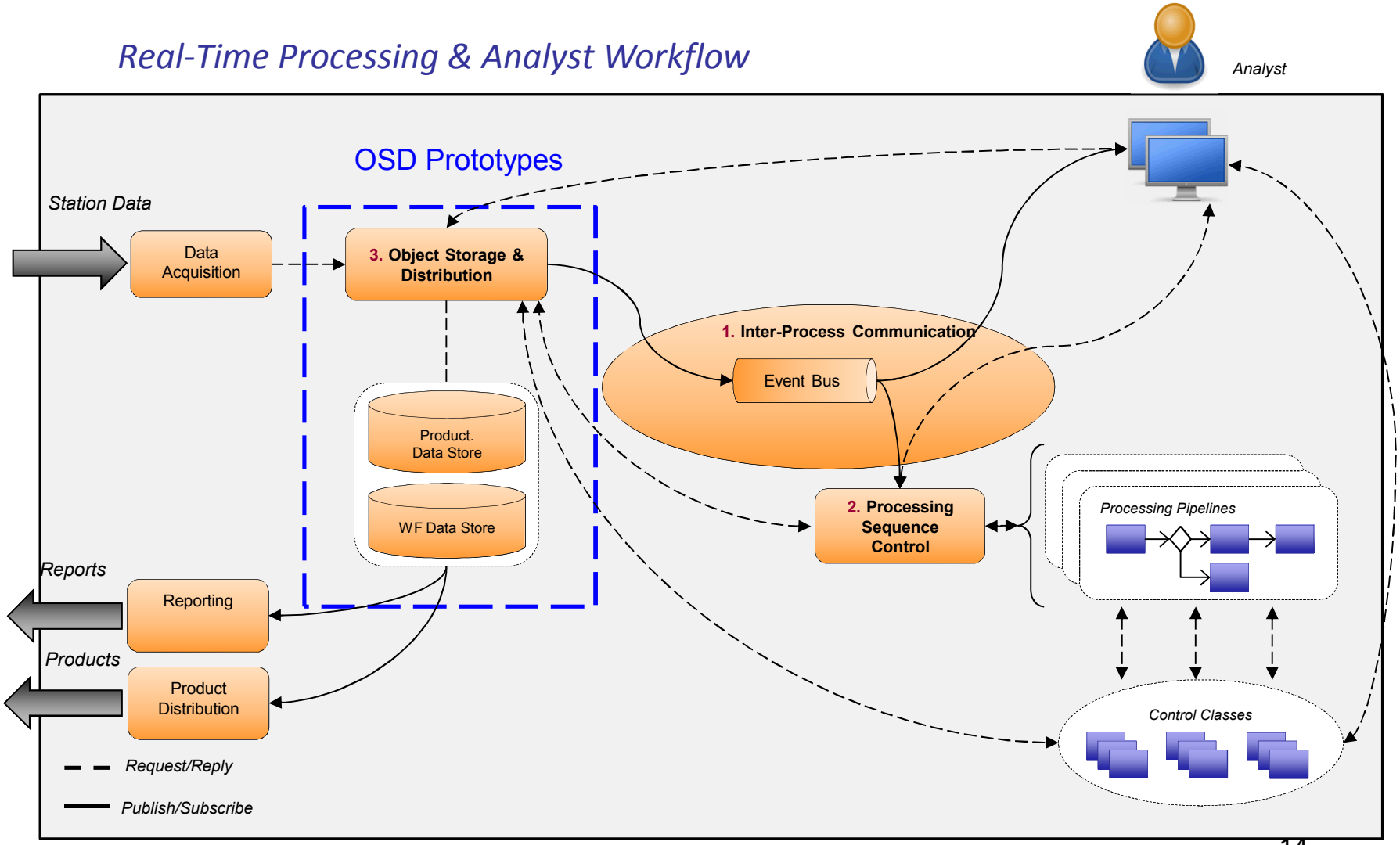
Sandia National Laboratories

- Selected PostgreSQL RDBMS for the storage of structured data within prototypes
  - Rationale: Widely used, advanced, open-source relational DBMS with high-quality tooling, documentation and community support
  - Note: the data storage technologies will likely continue to evolve during development (see Event Sourcing slides). The OSD abstraction layer has been defined to minimize the impact of this process
- Waveform storage design is an open trade
  - Prototyped WF storage using flat files, direct DB storage and Riak TS key-value store
- Selected Java Persistence API (JPA) for application-level RDBMS access
  - Rationale: Widely-used Object Relational Mapping (ORM) technology for Java
- Selected HTTP/JSON for request/reply COI APIs
  - Rationale: Ubiquitous standards enabling cross language/tech. stack CRUD access to stored data
- Selected RabbitMQ AMQP/JSON for publish/subscribe distribution of stored data
  - Rationale: Widely-used pub/sub standard (AMQP) and high-performance, cross-language, open-source implementation (RabbitMQ)
- Note: Selection of waveform encoding is an open trade (see IPC Follow-On Work slide)

Object Storage & Distribution – Prototype Architecture

Sandia National Laboratories

Real-Time Processing & Analyst Workflow



Object Storage & Distribution – Elaboration Prototypes

Sandia National Laboratories

- Implemented data storage (PostgreSQL) with initial COI APIs (JPA, HTTP/JSON) for select elements of the data model
  - Creation, retrieval & modification of stations, channels, waveforms, signal detections, signal detection hypotheses, feature measurements, events & event hypotheses
- Implemented subscription-based notification of data creation/modification events (RabbitMQ/JSON) for select elements of the data model as part of the Event Sourcing prototype
  - Signal detections, signal detection hypotheses, events, event hypotheses
- Implemented dedicated waveform storage using Riak key/value store (distributed, scalable alternative to flat files)
- Implemented alternate data storage solution using MongoDB as part of the Event Sourcing prototype
- Implemented data distribution to the web User Interface prototype using websockets

Object Storage & Distribution – Follow-On Prototyping Work

Sandia National Laboratories

- Waveform Storage
  - Select a technology for distributed, scalable waveform data storage
  - Candidates under consideration: InfluxDB, Riak TS, Apache Cassandra, etc.
- Data Model REST API Design
  - Select among REST API design options for HTTP/JSON access to data entities
  - Candidates: HATEOAS-enabled REST APIs, JPA-like REST APIs

Processing Sequence Control - Overview

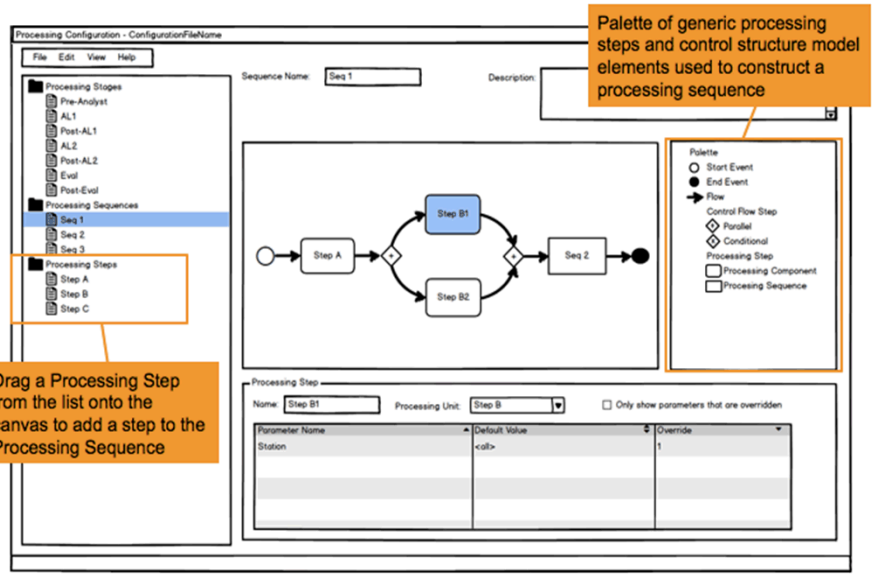
Sandia National Laboratories

- The Processing Sequence Control mechanism provides runtime orchestration of automated and interactive processing components within the System
- Specifically, the PSC provides for the definition, configuration, execution and status monitoring of processing sequences
- Processing Sequences are runtime arrangements of processing components that together support a higher-level activity within the system (e.g. station processing, network processing)

Processing Control Framework – Design Drivers

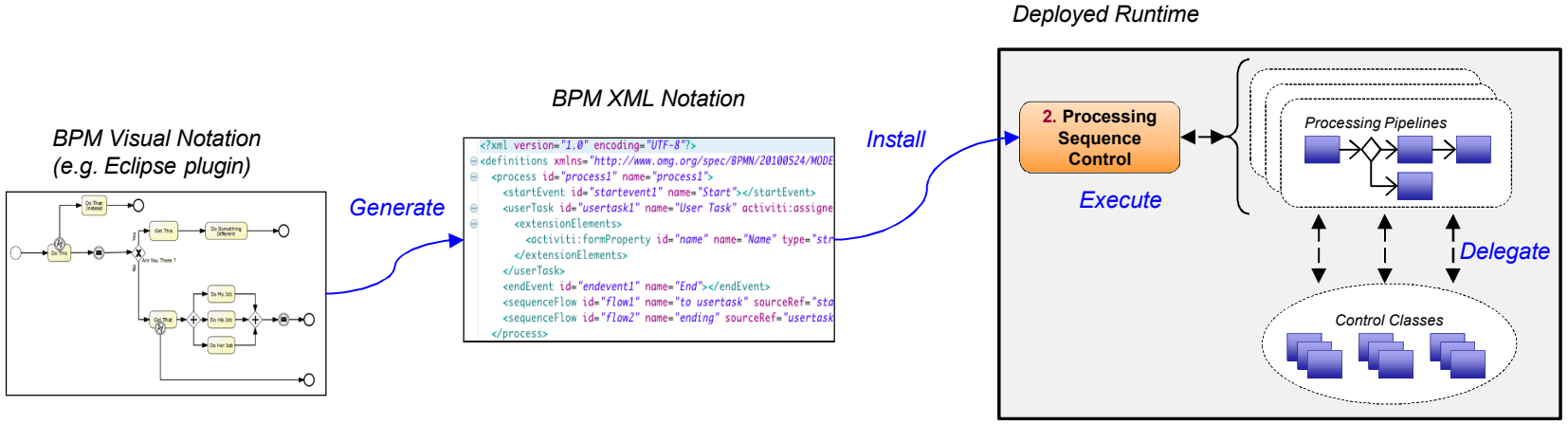
- Support a distributed/replicated processing components for scalability
- **Support definition of processing sequences, including sophisticated control structures**
  - Declarative and (preferably) visual notation
  - Iterative & conditional processing, parallel processing (e.g. forks and joins), invocation of nested and externally-defined sequences
- Support integration of new processing components (e.g. algorithm implementations)
- Support processing components implemented in languages selected for the re-engineered IDC system
- Support distributed processing

From ***Defines Processing Sequence*** UI Storyboard



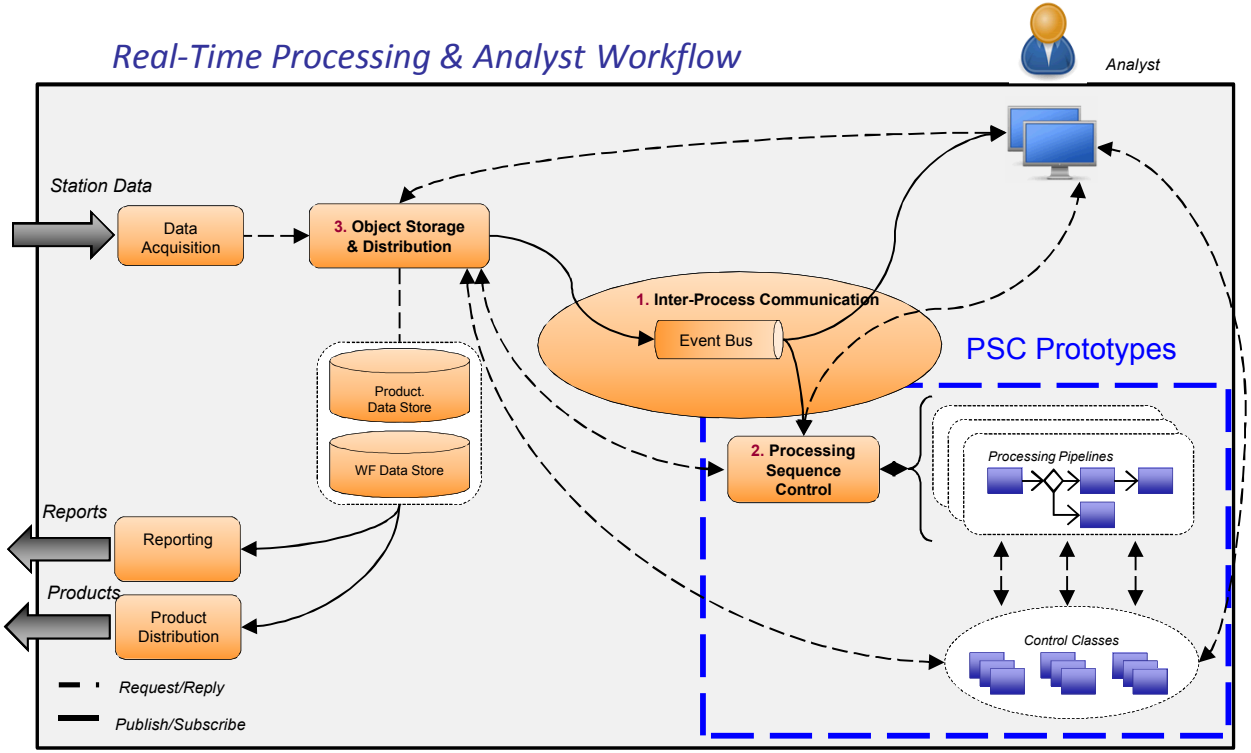
Processing Sequence Control – Technology Selection

- Evaluated several technologies, including Business Process Modeling (BPM) and rule engines as well as Batch Processing frameworks
- Selected Activiti BPM engine
  - Rationale:
    - Provides declarative Support definition of processing sequences with XML and visual notation (BPMN 2.0)
    - High-performance, scalable, open-source orchestration engine
    - Notation and engine support sophisticated control structures
      - Iterative & conditional processing, parallel processing (e.g. forks and joins), invocation of nested and externally-defined sequences



Processing Sequence Control (PSC) Prototyping

- Implemented a prototype PSC engine using Activiti BMP
- Defined & executed prototype processing sequences for station processing (QC processing, signal enhancement, signal detection, feature measurement)
  - Implemented stubbed Control classes for each step in the processing sequence invoked by the PSC engine via HTTP/JSON



Processing Sequence Control – Follow-On Prototyping Work

Sandia National Laboratories

- Assess and tune PSC engine performance for 1-10K channel processing on clustered testbed
- Evaluate emergent alternatives as needed (e.g. Spring Flo/Cloud Data Flo)
- Explore built-in Activiti rule engine as complementary orchestration approach

UI Framework – Overview

Sandia National Laboratories

- The User Interface Framework provides a software platform for development, deployment, configuration, layout & user customization of displays
- Prototyping effort focused on technology evaluation & selection
- Technology Options
  - Rich-client Java application (e.g. Eclipse, NetBeans)
    - \+ Supports user-customizable layout and display synchronization through RCP (e.g., NetBeans)
    - \- Design options are limited by standard Java look and feel
  - Browser-deployed web application
    - \+ More design freedom
    - \+ More momentum in industry
    - \+ Simplified deployment and upgrade model
    - \- Browser deployment limits some layout customizations
  - Natively-deployed web application (e.g. Electron)
    - \+ Leverages the same web technologies as browser-based deployment
    - \+ Easier to achieve the capabilities of a traditional desktop application
    - \- Requires more traditional deployment and upgrade model

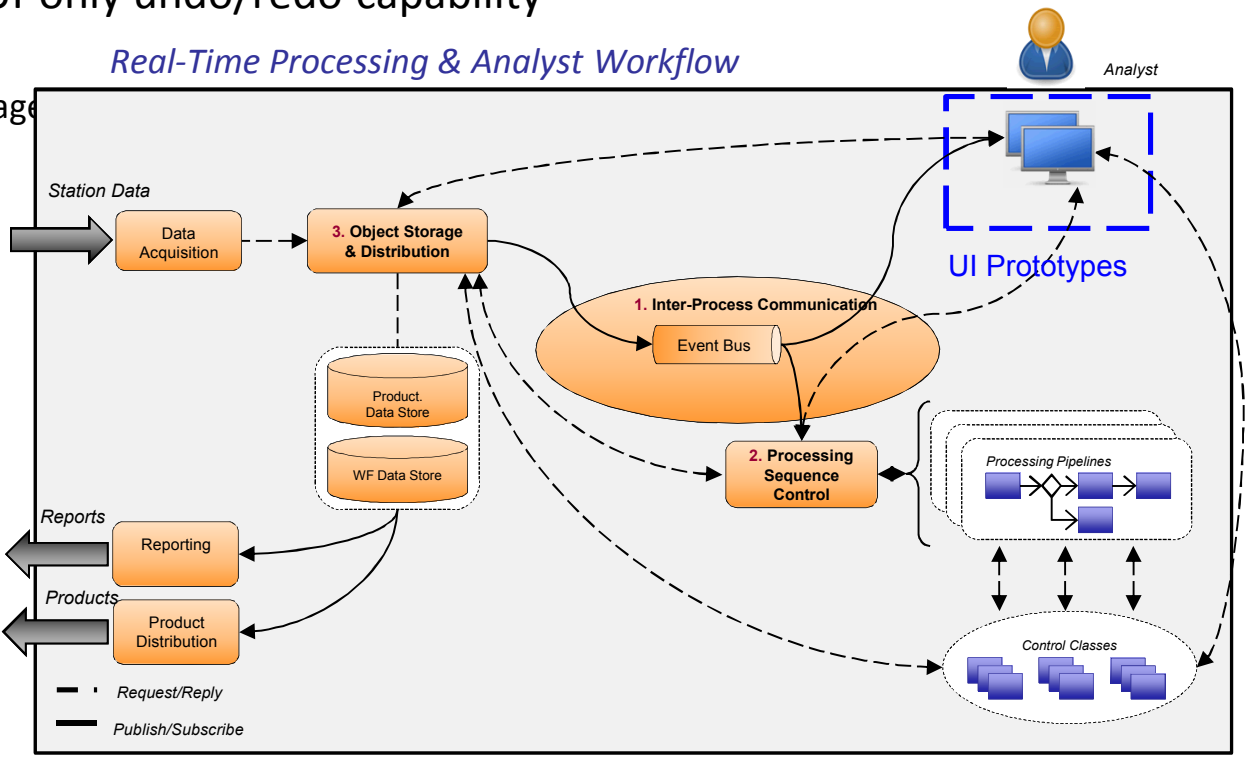
UI Framework – Technology Selection

Sandia National Laboratories

- Evaluated Java-based Rich Client Platforms (Eclipse, NetBeans) as well as web technologies addressing IDC design drivers:
  - User customization (layout, hot keys, etc.)
  - Responsiveness
  - Desire for robust, modern technologies
- Selected web-based UI design & technologies
  - Core UI programming languages will be JavaScript, HTML, and CSS
  - Deployment model likely will be a combination of browser-based and native deployment
    - Native deployment to support core analysis tasks
    - Browser-based deployment, as appropriate, to support other use cases (e.g., standalone subsystem, external users, system control…)
  - Rationale:
    - Widely-used, robust, mature technology ecosystem with demonstrated capabilities addressing:
      - Interactive performance
      - User workspace customization
      - Ease of deployment and upgrades
      - Integration of GIS capabilities
      - Scalability, maintainability of the code base
  - See Backup slides for additional details on the web UI technology evaluation

UI Framework – Prototyping

- Implemented basic Java RCP waveform display prototypes
- Implemented Browser-based and standalone web-technology Analyst display prototypes, including:
  - Waveform visualization, signal detection picking, feature measurement (continuous FK), basic map display of stations & events
- Implemented an initial UI-only undo/redo capability
  - Single global undo stack
  - No server-side state manage



UI Framework – Follow-On Prototyping Work

Sandia National Laboratories

- Data Communication
  - Select technologies for data communication with the web-based user interface, providing:
    - Efficient data access APIs between browser and backend
    - Data subscriptions for dynamic display updates
  - Candidates: GraphQL, Falcor, Websockets
- Undo/Redo accounting for server state (processing context)
- GIS technology selection

Sandia National Laboratories

- Prototyping Overview
- Hybrid Architecture Context
- Prototyping Focus
- Software Mechanism Prototypes
  - Technology Summary
  1. Inter-Process Communication
  2. Processing Sequence Control
  3. Object Storage & Distribution
  4. User Interface Framework
- **Key Feature Prototypes**
  - Provenance – Event Sourcing
- Summary

KEY FEATURES - PROVENANCE

Provenance – Overview

- The IDC SRD includes requirements to preserve the history of the System state
- Motivated by the desire to track the provenance of information
  - *SRD-139: The system shall preserve a complete history/audit trail for every saved event and allow users to recall and review an event at any point in its history.*
  - *SRD-189: The system shall use the data management system to store and retrieve automatic and interactive processing parameters and results for use by subsequent processing.*
  - *SRD-233: The system shall collect, display, and store waveform data availability at specific points of time. NOTE: The intent of this requirement is to allow the display of the exact data availability at any point in the processing history for either automatic or interactive processing.*

Provenance – Event Sourcing + CQRS

Sandia National Laboratories

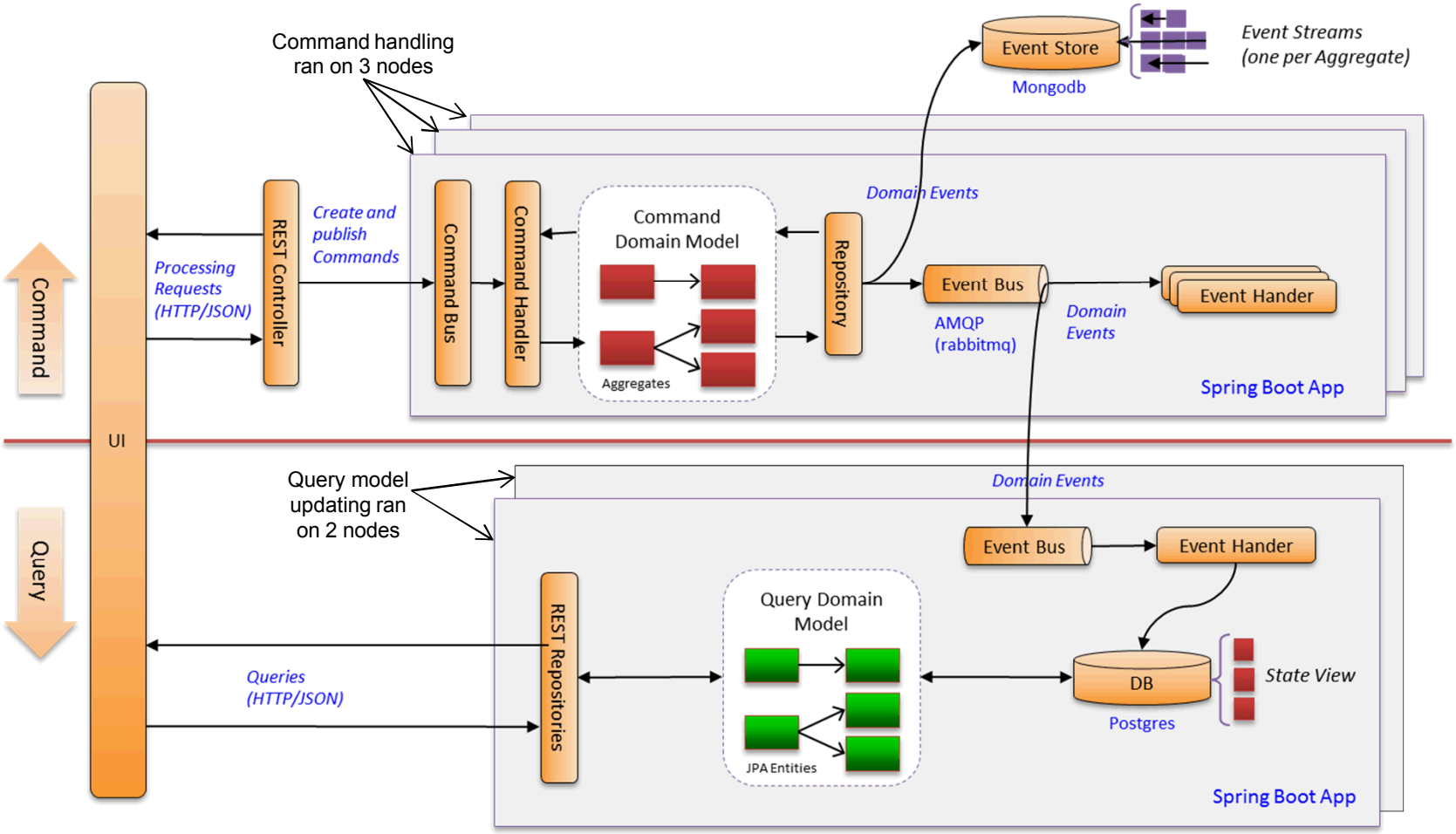
- Event Sourcing (ES) is an architectural approach that seeks to address this type of requirement
  - Generalized solution to track object's historic states
  - Also allows rehydrating those states
  - Typically seen in industries where:
    - Rigorous audit logging is a critical requirement and/or
    - State is naturally defined as the sum of a sequence of events
    - E.g. Banking, stock trading
- Supporting queries is a problem for Event Sourcing
  - The Event Log does not provide a view into current state
  - Replaying the Event Log in for each query likely is not practical
- Command Query Responsibility Segregation (CQRS) addresses this problem
  - An architecture pattern where the System uses separate models for modifying and querying state
  - In an event sourcing system, a separate cache or DB instance provides a view of System state for query purposes that is populated from the stream of domain events published via an event bus
  - Any number of query models can be maintained for different purposes

Provenance – Event Sourcing + CQRS Prototype

Sandia National Laboratories

- Developed an initial ES + CQRS prototype in E6
- Goals:
  - Investigate technologies
  - Investigate data model implications
  - Demonstrate ES design for sample Commands affecting event hypotheses (create, associate signal)
- Selected Technologies
  - Axon framework (Java ES + CQRS distributed computing framework)
  - Spring Boot (Java application development framework)
  - RabbitMQ (event bus implementation supporting distributed system)
  - MongoDB (event store implementation)
  - JGroups (distributed Command bus; only option available in Axon 2.4)

Provenance – Event Sourcing + CQRS Prototype Design



Findings

Sandia National Laboratories

- ES + CQRS
  - Appears to be a viable approach to recording and recreating state history in the reengineered system
    - All state changes recorded (i.e., those already known to be useful, those that might be useful in the future)
  - May support some or all undo/redo requirements
  - Command based processing could be leverage to log and replay operations
  - Adds complexity to the architecture
    - Separate data models, separate domain event store and query DB, snapshotting, playback)
- There are few OSS frameworks providing ES/CQRS support
- Axon
  - Is fairly easy to use, but there are limited code examples to draw on
  - Under active development
  - Configuring Axon distributed processing was the most challenging step
    - Need to resolve locking / concurrent writers issues (not Axon specific)

Follow-On Work

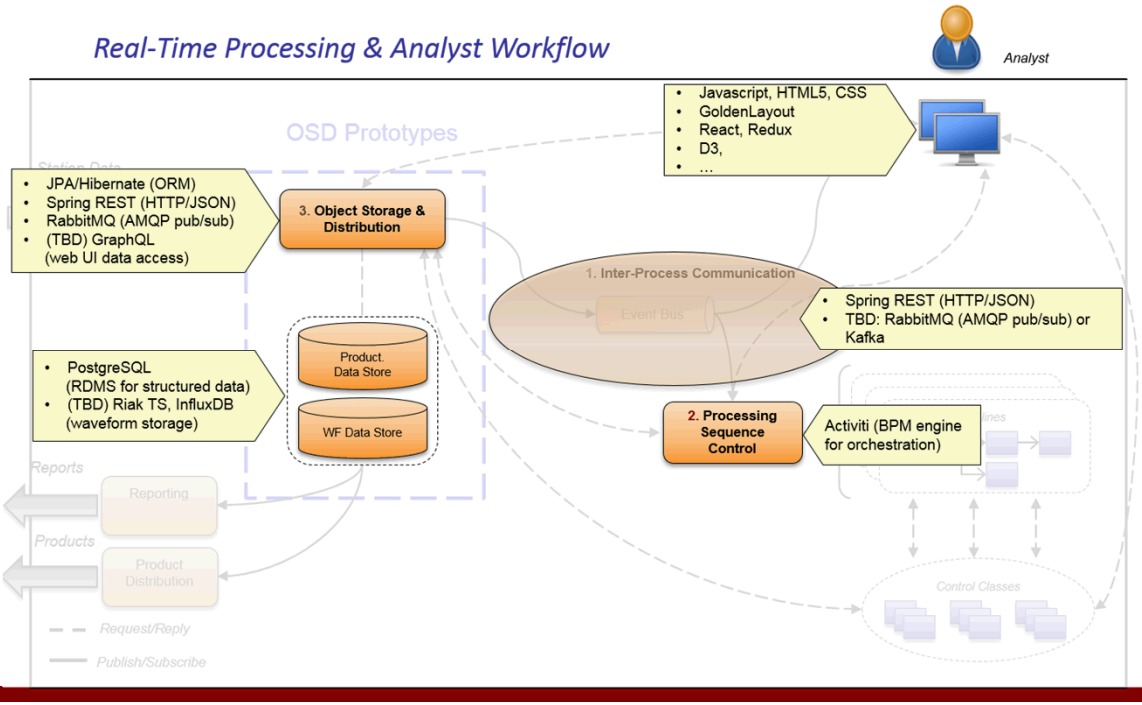
- Elaborate the command-side data model to identify the events and aggregates (development activity)
- Estimate scaling needs of the domain event store, command handlers, query models based on predicted monitoring network size
- Determine if undo-redo should be based on event sourcing or a snapshotting policy

Sandia National Laboratories

See the Event Sourcing white paper for further details

Summary

- Implemented prototypes for key mechanisms, including
  1. Inter-Process Communication
  2. Processing Sequence Control
  3. Object Storage & Distribution
  4. User Interface Framework
- Selected technologies and initial COTS for most mechanism components
- Implemented an Event Sourcing prototype as a candidate solution for provenance



Sandia National Laboratories

BACKUP

Sandia National Laboratories

WEB UI TECHNOLOGY EVALUATION

UI technology stack options

Sandia National Laboratories

1. Rich-client Java application
   - \+ Supports user-customizable layout and display synchronization through RCP (e.g., NetBeans)
   - \- Design options are limited by standard Java look and feel
2. Browser-deployed web application
   - \+ More design freedom
   - \+ More momentum in industry
   - \+ Simplified deployment and upgrade model
   - \- Browser deployment limits some layout customizations
3. Natively-deployed web application
   - \+ Leverages the same web technologies as browser-based deployment
   - \+ Easier to achieve the capabilities of a traditional desktop application
   - \- Requires more traditional deployment and upgrade model

Proposed path forward

Sandia National Laboratories

- System user interfaces will be developed using web technologies
- Core UI programming languages will be JavaScript, HTML, and CSS
- Deployment model likely will be a combination of browser-based and native deployment
  - Native deployment to support core analysis tasks
  - Browser-based deployment, as appropriate, to support other use cases (e.g., standalone subsystem, external users, system control…)
- Common UI code will be reused across deployments

Decision criteria

- Interactive performance
- Ability to save and restore user-defined workspaces
- Ease of deployment and upgrades
- Integration of GIS capabilities
- Scalability, complexity, and maintainability of the code base

Interactive performance

Sandia National Laboratories

- The UI must provide acceptable performance under both typical and stressing conditions
- Loading and interacting with large amounts of waveform data
  - Prototype loads ~600 channels, 2 hours of data per channel, in ~1 minute over a 1Gbps connection, using a 64-bit build of a modern browser (e.g., Firefox, Chrome)
  - Interaction (e.g., panning, zooming, redrawing) is responsive using custom WebGL rendering code
  - Optimizations (e.g., background loading, serialization/compression methods) will improve both load times and interaction response times
- Synchronizing data across multiple complex displays
  - Prototype demonstrates synchronization across 12+ displays
  - Uses Golden Layout to display multiple tabs in a single browser window
  - Uses a combination of React JS and Redux to manage application state

React + Redux for state management

Sandia National Laboratories

- **React**: A JavaScript library for building performant UIs
- **Redux**: A predictable state container for JavaScript applications

- Simplified state management for complex JavaScript applications
  - The state of the entire application is stored in an object tree within a single store
  - Data flows in one direction – from a central dispatcher to the store to the views
  - State is read-only – changes are made via pure functions that take the previous state and an action and return the next state
  - Views (i.e., React components) listen for change events, then retrieve new data from the store and provide it to all of their child views
  - React components re-render minimal parts of the DOM to satisfy new data

Ability to save and restore user-defined workspaces

Sandia National Laboratories

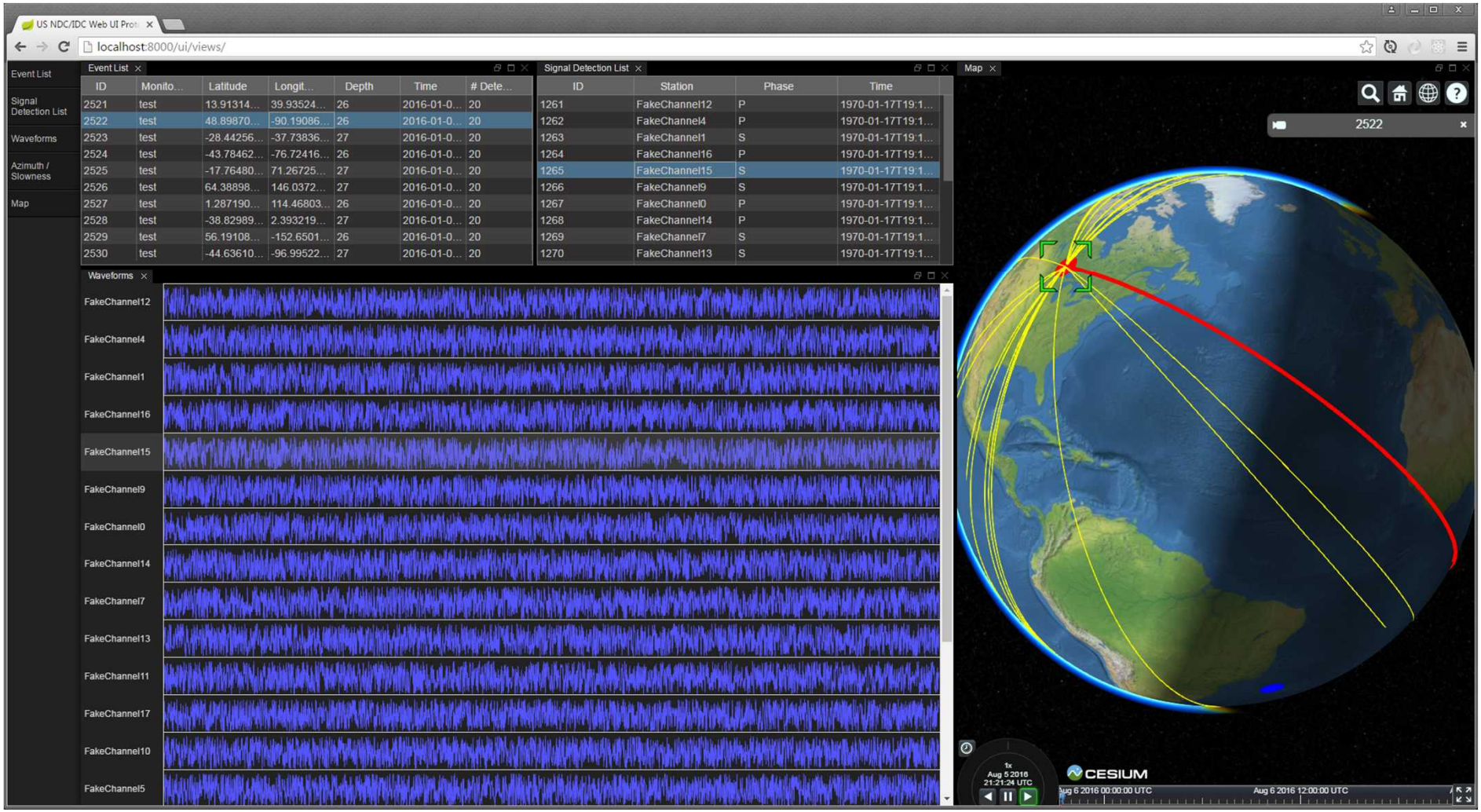
- System users need the ability to organize displays into custom workspaces that can be saved and restored across sessions
- Prototype uses Golden Layout, a JavaScript library that supports opening, closing, dragging/dropping, resizing, and docking/undocking individual tabbed displays within a single browser window
  - Golden Layout provides proof of concept, but will require some refactoring to fully support our use cases
- Browser limitations prevent full restoration of layouts across multiple screens
  - Can be mitigated by deploying natively using a framework like Electron

Ease of deployment and upgrades

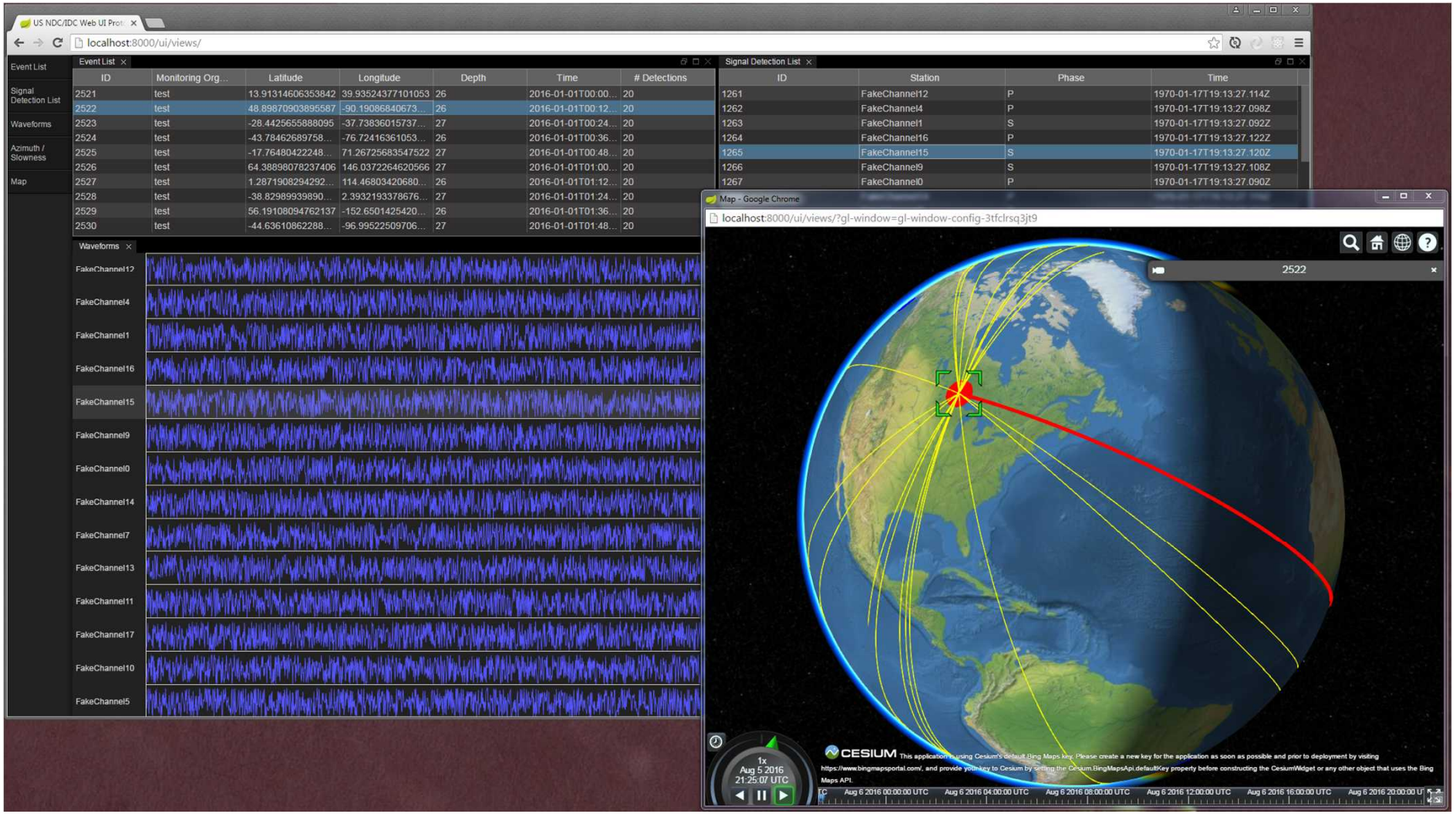
Sandia National Laboratories

- Accessibility and ease of deployment are key benefits of a browser-based web UI
  - No installation on individual workstations
  - Upgrades are deployed once to a server
- However, this model is limited to a browser-based deployment, and benefits would be diminished for…
  - Users without network connections (e.g., on a standalone system)
  - Cases where web services need to be forward-deployed to meet performance needs
- A hybrid deployment model would allow core users (e.g., analysts) a better user experience via a natively deployed application while giving other users the flexibility of accessing the application via a browser

Multi-display layout with Golden Layout

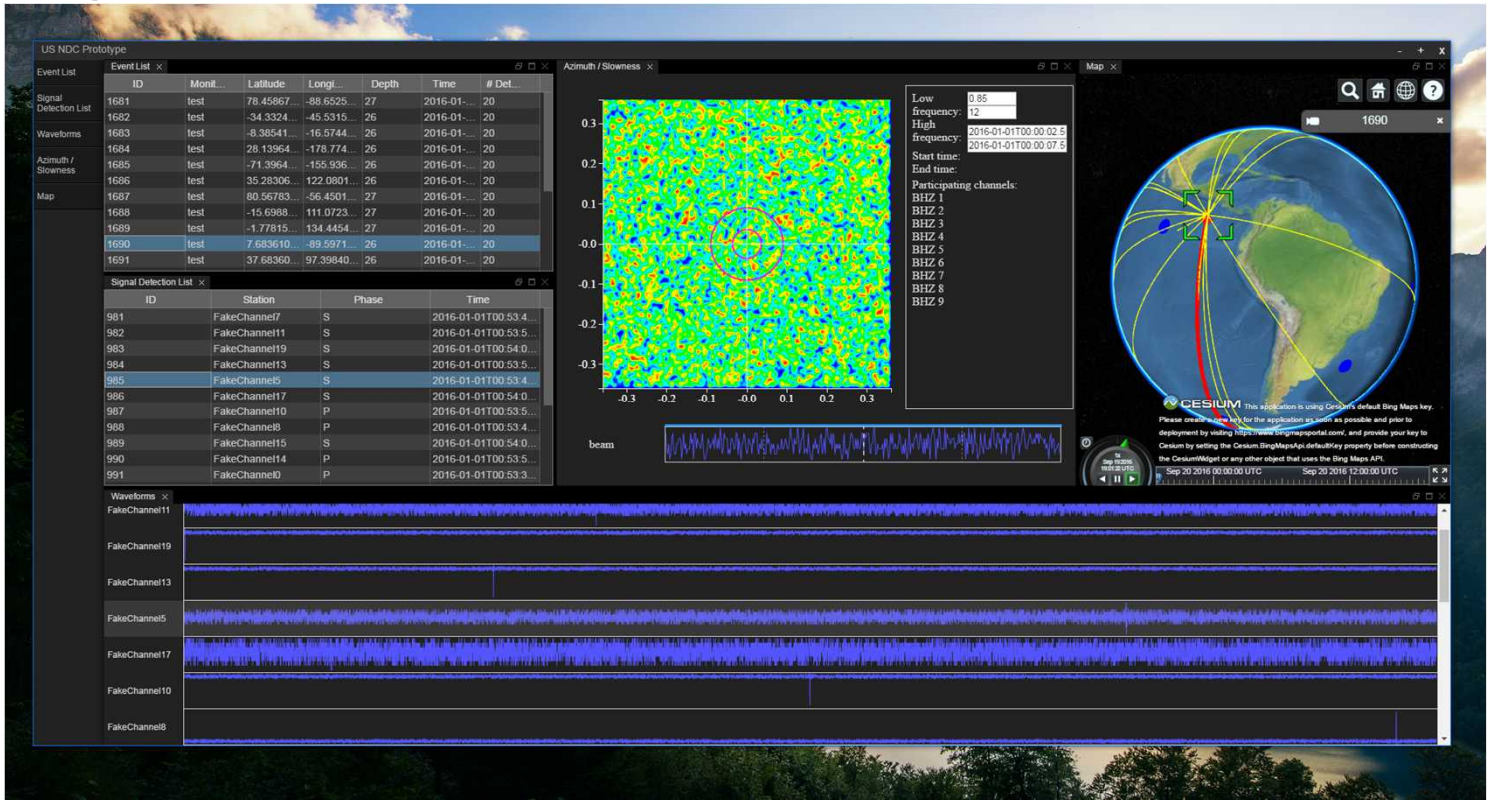


Undocking displays with Golden Layout



Native deployment with Electron

- **Electron**: A framework for creating native applications with web technologies like JavaScript, HTML, and CSS
- Allows integration with mechanisms provided by the underlying OS (e.g., menus, notifications, and shortcuts)



Integration of GIS capabilities

Sandia National Laboratories

- Goal is to provide robust GIS capabilities that are well-integrated with the rest of the application
- Industry leaders (e.g., Google, ESRI, AGI) are moving toward web-based tools
  - Currently, there is more support for 2D rather than 3D projections
- Our web-based GIS technology stack could include a combination of open source tools (e.g., GeoServer, Cesium, OpenLayers) or could be implemented using the ArcGIS platform
- Regardless of the chosen technology stack, GIS capabilities will be implemented based on OGC standards (e.g., Web Feature Service, Web Map Service, Web Coverage Service)

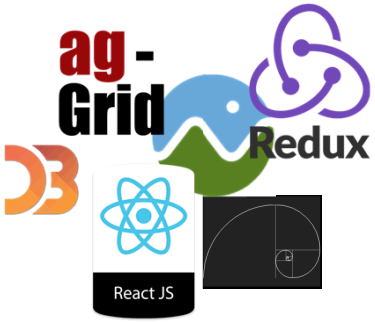
Scalability, complexity, and maintainability of the code base

Sandia National Laboratories

- Leveraging third-party tools is beneficial, but adds risks because web technologies can appear, change, and disappear rapidly
  - As a best practice, design a web-based UI architecture assuming that some number of third-party tools will eventually need to be replaced
  - E.g., create a thin view layer, isolate view code from application logic, define generic APIs that are agnostic to third-party implementations
- In terms of code complexity, a Java implementation is less risky due to its object-oriented nature and well-established patterns
  - JavaScript, as a dynamic, untyped language, adds risk in terms of complexity, scalability, and maintainability of large-scale code bases
  - TypeScript, a typed superset of JavaScript developed by Microsoft, helps mitigate this problem with optional types – allowing for practices such as interface definition, static checking, and code refactoring

UI technology interaction

Define project structure & library dependencies



Write UI in TypeScript using libraries

Transpile & bundle TypeScript code into a single JS file

HTML page loads bundled JS, UI is initialized

Browser requests page using HTTP GET

References

- **ag-Grid**: A data grid library that provides robust capabilities for the display of tabular data. Provided an initial test case for wrapping third-party tools for easy replacement in the future. (https://www.ag-grid.com)
- **Cesium**: An open-source 2D/3D virtual globe implemented with WebGL. Developed by AGI. Leading candidate for web-based map visualization because it is one of only a few options that currently support 3D rendering. (https://cesiumjs.org/)
- **Electron**: A framework for creating native applications using web technologies. A likely candidate for deployment for scenarios where browser-based deployment is not viable. (http://electron.atom.io/)
- **Golden Layout**: A multi-window layout manager for web applications. Supports RCP-like look and feel inside the browser; however, the library will require rework to fully support our robust use-case. (https://www.golden-layout.com/)
- **React**: Supports efficient update and rendering of UI components based on state. Developed by Facebook and used widely in industry. A likely candidate for the foundational framework of the System UI. (https://facebook.github.io/react/)
- **Redux**: A library for managing complex application state. Used for initial investigation of undo/redo capabilities in the UI. (http://redux.js.org)
- **Typescript**: A typed superset of JavaScript developed by Microsoft. A likely candidate for use since it has already made our relatively small UI prototyping code base easier to manage. (https://www.typescriptlang.org/)