eMontage: An Architecture for Rapid Integration of Situational Awareness Data at the Edge

Soumya Simanta
Gene Cahill
Ed Morris
eMontage: An Architecture for Rapid Integration of Situational Awareness Data at the Edge
Motivation – Situational Awareness

First responders and others operating in the “last mile” of crisis and hostile environments are already making use of handheld mobile devices in the field to support their missions.

Rapid Incorporation of New Data Sources
• Many data sources (real-time, historical, …)
• Data is fragmented across different apps on the mobile device

Minimized Information Overload
• Edge users are under high cognitive load
• Information required is a function of user’s context and therefore dynamic

Simple Use
• Users are under high stress
• Small screen devices

Resource Constrained Hostile Environment
Hostile Environments Characteristics

Wimpy edge nodes
- Limited resources (CPU, battery and memory) on mobile nodes
- Example: Expensive computations on a smartphone may drain the battery fast

Limited or no end-to-end network connectivity
- Implicit assumption of WAN connectivity is not always valid
- Example: No access to internet during a disaster, DoS attack

High cognitive load
- Application latency and fidelity become important
- Example: A slow application will increase the cognitive load on the user

Bounded elasticity
- Upper bound on number of consumers known in advance
- Example: Fixed number of first responders in a location

Dynamic environment
- Static deployment topologies cannot be assumed; Survivability essential
- Example: An automobile with a server may not be available
Context Diagram

Geo-location based Remote Data Sources

Private
- Source 1
- Source 2

Public
- Twitter
- Google Places
- FourSquare
- Earthquake DB

Enterprise Domain

Tactical Domain

eMONTAGE Server

Mobile Client 1
Mobile Client 2
Mobile Client 3

Real-time data
- User-defined Data Filters
- User-defined Map Customizations
Architecturally Significant Requirements

Extensibility
- Add new data source quickly with minimal impact on existing sources

Runtime Configurability
- Make data sources user configurable (e.g., using data filters) at runtime

Performance
- Minimize network bandwidth usage of the tactical network

Energy Efficiency
- Optimize energy consumption on mobile handheld devices

Usability
- Provide a responsive and unified user interface

Availability
- Support intermediate disconnections with remote data sources

Security
- Support existing security protocols and provide transport layer security
Runtime C&C View

- **eMontage Mobile Client**
  - Map View
  - Search View
  - Rule Builder
  - REST Handler
  - Spring REST Android Client

- **eMontage Server**
  - Jetty App Server
  - Camel Runtime
    - Route_1
    - Route_2
    - ... Route_n
  - Producer Template
  - eMontage Controller
  - Dispatcher Servlet
  - Spring MVC Runtime

**Legend**
- System Boundary
- Custom Runtime Component
- 3rd Party Runtime Component
- File
- Call Return
- File Read/Write
- HTTP Connector

Software Engineering Institute
Carnegie Mellon

SATURN 2013
© 2013 Carnegie Mellon University
Request Response Interaction

Android Client — Dispatcher Servlet — Spring MVC Controller — Camel Producer Template — Camel Route — Rule Engine — Remote Data Service

REST Request with parameters and rules

Request params and rules

Invoke Camel route

Pass parameters and rules

Parameters to invoke remote service

Data set

Data set objects

Filtered objects

Filtered objects

JSON Response
Publish Subscribe Interaction

Android Client | Dispatcher Servlet | Spring MVC Controller | Remote Data Service

REST Subscribe Request → Request params and rules → Create Camel Route → Data Message

Mobile Client Network Thread - Data messages - Transform Messages
Example Routes

**Google Place Route**

1. **S**: Process HTTP Parameters from input message and set HTTP headers
2. **Set query string headers**
3. **REST GET request**
4. **Invoke web service**
5. **Unmarshal results using JSON parsers to custom object**
6. **Split results into individual objects**
7. **Pass objects to embedded drools rule engine**
8. **Send filtered objects from Drools channel back to caller.**

**National Weather Service Alerts Route**

1. **S**: Process HTTP Parameters from input message and set HTTP headers
2. **Use RecipientList to set parameters in the HTTP URL**
3. **Invoke web URL**
4. **Parse log to remove unwanted text and create well formed XML document**
5. **Parse XML into a single Java XML Document object.**
6. **Split XML Document into domain specific Java object**
7. **Send filtered objects from Drools channel back to caller.**
Extensibility – Adding New Data Sources

Problem
• New data sources are available in the field
• Adding and validating them is time consuming

Assumptions
• The data source has a remote API
• The data format is defined and stable

Solution
• Minimize coupling between data sources by encapsulating each data source
• Implement common connectors (request/response and publish subscribe)

Future extensions
Automate common tactical integration patterns to provide an end-user programming interface
Data Model

Data model is represented as objects (POJOs) shared between clients and server.

Data model changes must be synchronized between clients and servers.

- Our assumption: data model is “relatively” stable.

Data model can be created in the following ways:

- **Manual definition** – works best in case of a simple data model.
- **Code generation** – WSDL2Java to generate code.
- **Reuse existing library** – Twitter4J is an existing Java implementation of Twitter API.
Mashup Mechanism

Merging data across data models is a mechanism to relate data across models.

• Example: foreign keys in a relational database

In eMontage, we assume a large proportion of situational awareness data has some form of geo-location associated with it

• use geo-location as the common key

All data is currently mashed up on a map-based interface.

• as long as two data elements from different data sources are referenced by geo-location (latitude and longitude), they will always be displayed correctly on a map

• the actual relating of information will happen with the user
## Example Data Sources

<table>
<thead>
<tr>
<th>Data Source Name</th>
<th>Data Format</th>
<th>Wire Protocol</th>
<th>Security Mechanisms</th>
<th>Data Model Complexity</th>
<th>User Interface Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Google Places</td>
<td>JSON</td>
<td>REST</td>
<td>Token-based</td>
<td>Low</td>
<td>Geo-points on map</td>
</tr>
<tr>
<td>Twitter</td>
<td>JSON</td>
<td>REST</td>
<td>Token-based</td>
<td>High</td>
<td>Geo-points on map</td>
</tr>
<tr>
<td>FourSquare</td>
<td>JSON</td>
<td>REST</td>
<td>Token-based</td>
<td>High</td>
<td>Geo-points on map</td>
</tr>
<tr>
<td>Private Data source 1 (real-time)</td>
<td>XML</td>
<td>UDP</td>
<td>No security mechanism</td>
<td>Low</td>
<td>Polygons, Geo-points on map</td>
</tr>
<tr>
<td>Private data source 2 (historical)</td>
<td>SOAP</td>
<td>HTTP</td>
<td>Custom</td>
<td>High</td>
<td>Geo-points on map</td>
</tr>
<tr>
<td>National Weather Service Alerts</td>
<td>Custom log files</td>
<td>HTTP</td>
<td>No security mechanism</td>
<td>Medium</td>
<td>Polygons</td>
</tr>
</tbody>
</table>
# Configurability- User-defined Runtime Filtering

<table>
<thead>
<tr>
<th>Problem</th>
<th>Information overload</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assumption</td>
<td>The user knows what information they need in a particular context (e.g., location, keywords, date ranges)</td>
</tr>
</tbody>
</table>
| Solution         | Provide mechanisms that allow users to reduce the volume of information  
|                  | • using rule-based filtering at runtime |
| Future extensions| Provide “data discovery” mechanisms (e.g., visualizations, clusters, outliers) when the user does not know what information they need |
## Usability - Unified User Interface

<table>
<thead>
<tr>
<th>Problem</th>
<th>Data is fragmented across multiple applications and databases</th>
</tr>
</thead>
</table>
| Assumption | • Data is geo-coded  
• A unified view provides more value compared to isolated views of data from different sources |
| Solution | Mashup of geo-code data viewed on a map allows visual unification of data |
| Future extensions | • Provide other non-map based visualizations  
• Provide data join mechanisms |
Performance - Minimized Bandwidth Utilization

Problem: Bandwidth is a scare resource at the “last mile” of edge

Assumption: Possible to have an intermediary node in the network

Solution: Add an intermediary node
- Use filtering at the source (only send information is required by the mobile nodes)
- Transform to a more bandwidth optimized format (e.g., XML to JSON/Protocol Buffer)

Future extensions: Use protocol transformation (use a SPDY instead of HTTP)
Power Consumption - Offloading Expensive Computation

Problem: Mobile nodes have limited resources (CPU, battery and memory)

Assumption: Possible to have an intermediary node in the network

Solution: Perform expensive computation (e.g., XML parsing, multiple network calls) on a proximate, relatively resource rich node

Future extensions: Use multi-node cloudlets to increase performance and fault tolerance
## Availability - Disconnected Operations

<table>
<thead>
<tr>
<th>Problem</th>
<th>Edge nodes may have to work in disconnected or semi-connected mode (from enterprise/TOC network)</th>
</tr>
</thead>
</table>
| Assumption | • Possible to deploy a resource-rich node locally (e.g., on an automobile)  
• Real-time data is generated *locally*  
• Possible to know in advance what data will be required for a mission (e.g., maps by locations) |
| Solution | Localize and cache data sources on a cloudlet. |
| Future extensions | Use persistent distributed caching.  
Adaptive pre-fetching to support intermittent disconnections |
Architectural Alternatives

Native Mobile Client Only

• A **native, mobile client** app directly connected to backend data sources

Mobile Browser-Server

• A **mobile browser** client to a server that acts as an **intermediary** between the mobile client and the backend data sources

Native Mobile Client-Server

• A **native, mobile client** app connected to an in **intermediary** server
## Architectural Alternatives

<table>
<thead>
<tr>
<th>Feature</th>
<th>Native Mobile Client Only</th>
<th>Mobile Browser-Server</th>
<th>Native Mobile Client-Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reuse of COTS</td>
<td>Low</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>Protocol/Data format transformation</td>
<td>No</td>
<td>Yes</td>
<td>Maybe</td>
</tr>
<tr>
<td>Intermediate filtering</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Bandwidth optimization</td>
<td>No</td>
<td>Yes</td>
<td>Maybe</td>
</tr>
<tr>
<td>Disconnected operations</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Rich user interface</td>
<td>Yes</td>
<td>Yes</td>
<td>Maybe</td>
</tr>
<tr>
<td>Energy efficiency</td>
<td>No</td>
<td>Yes</td>
<td>Maybe</td>
</tr>
<tr>
<td>Fault Tolerance</td>
<td>High</td>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td>Caching</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Runtime modifiability</td>
<td>Low</td>
<td>High</td>
<td>High</td>
</tr>
</tbody>
</table>
Current and Future Work

More intuitive user interface
  • Support other views of data
  • Provide data exploration and discovery capabilities

Focus on performance
  • Use caching
  • Allow use of multiple processors/cores when possible

Add security mechanisms
  • Use with Wave Relay radios
  • Add transport and message level encryption

Integrate with edge analytics
  • Build edge analytics techniques on top of current eMontage implementation
Questions