US ARMY RDECOM

U.S. Army Research, Development and Engineering Command

ARL

TECHNOLOGY DRIVEN. WARFIGHTER FOCUSED.

Terra Harvest Mission Programming Approach

Jesse Kovach
Battlefield Information Processing Branch, ARL
24 April 2012

Overview

- Terra Harvest is a framework for developing interoperable UGS controllers
- Asset plugins (data producers) post observations (lines of bearing, images, etc) to a persistent store (database) using a common lexicon (Java classes generated from a set of XML schemas)
- Controllers need a way to “wire” different assets, communications devices, algorithms, and the like together to perform useful functions
  - “Mission logic” / “Mission programming” / “Trigger rules” / etc
  - **How to do this?**

Requirements

1. **Must*** provide an equivalent level of functionality to currently available UGS systems
2. **Must** be extensible to support new types and classes of devices, algorithms, etc. without requiring extensive re-engineering
3. **Must** perform well in a size, weight, and power constrained embedded environment

4. **Should*** be capable of accommodating missions of arbitrary complexity
5. **Should** allow for the creation of GUI tools to develop missions
6. **Should** be able to identify unsatisfiable/inconsistent/redundant/contradictory mission logic to warn users of potential programming errors

* See IETF RFC 2119

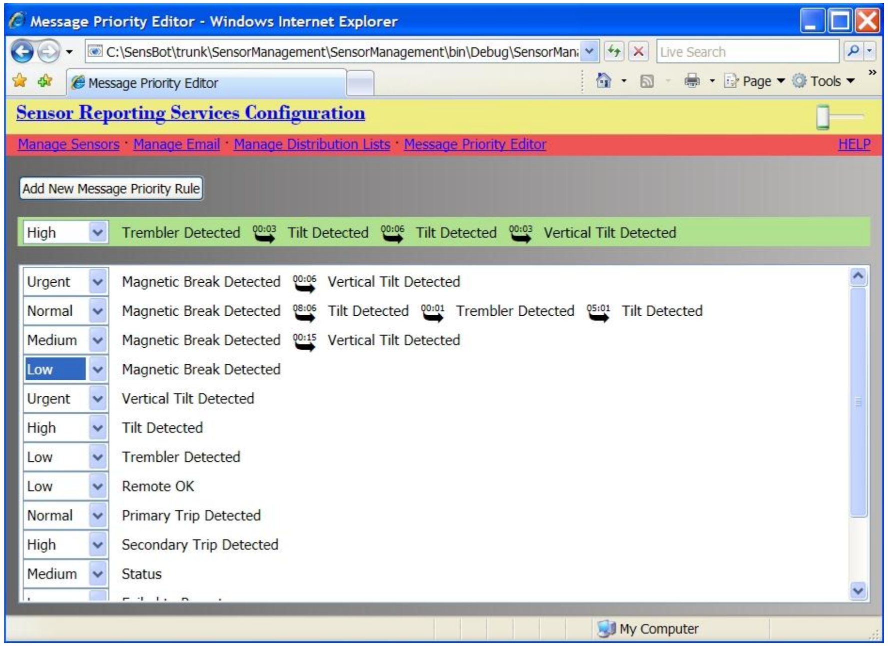
Possible Approaches

- Hardcoded Rules Engine
- SensorML Process Chains
- Web Ontology Language (OWL) / Semantic Web Rules Language (SWRL)
- SQL Triggers
- Scripting Languages

- Probably others…

Hardcoded Rules Engine

- Approach used by most current UGS systems
- Wires predefined actions together
- Easy to build GUIs, easy for users
- Not extensible



SensorML Process Chains

- OGC standard for describing (among other things) sensor process models
- Can be used to describe UGS missions, but it is a poor fit
  - SensorML describes a sequence of operations that produces a result (e.g. for a remote sensing system – capture image, scale it, apply a processing algorithm, then save it)
  - UGS missions need to describe a set of operations to take in response to events, which is awkward in SensorML
    - Difficult to model Boolean “or” conditions
- Does not convey semantic information well
  - Depends on external, non-standardized ontologies

OWL and SWRL

- W3C recommendations for describing ontologies (OWL) and rules that infer knowledge from ontologies (SWRL)
- Define ontologies for sensor system components and use rules to describe actions to take in response to events
- Easily extensible
- Existing open source and commercial processing engines and GUI tools
- Can identify inconsistent/impossible missions
- Way too slow!

```
<owl:NamedIndividual rdf:about="http://th-owl/EC10-M04.owl#EC10-M04">
  <rdf:type rdf:resource="&UGS;Mission"/>
  <UGS:hasMissionStartTime rdf:datatype="&xsd;dateTime">2010-06-
25T14:06:01</UGS:hasMissionStartTime>
  <UGS:hasRCGSID rdf:datatype="&xsd;integer">297</UGS:hasRCGSID>
  <UGS:hasComments rdf:datatype="&xsd;string">CAST1, CAST2,  PHX D1.  CAST
detections sends detection message and puts D1 in motion detection. Camera
detection takes picture and sends thumbnail. RCGS SOH one per
hour.</UGS:hasComments>
  <UGS:hasName rdf:datatype="&xsd;string">EC10-M04</UGS:hasName>
  <UGS:hasNumber rdf:datatype="&xsd;string">EC10-M04</UGS:hasNumber>
  <UGS:hasProcedure rdf:resource="http://th-owl/EC10-M04.owl#EIP1"/>
  <UGS:hasProcedure rdf:resource="http://th-owl/EC10-M04.owl#EIP2"/>
  <UGS:hasProcedure rdf:resource="http://th-owl/EC10-M04.owl#EIP3"/>
  <UGS:hasTransport rdf:resource="http://th-owl/EC10-M04.owl#Iridium"/>
  <UGS:hasCoordinates rdf:resource="http://th-owl/EC10-
M04.owl#MissionCoordinates"/>
  <UGS:hasRCGSMode rdf:resource="http://th-owl/EC10-M04.owl#MissionMode"/>
  <UGS:hasAsset rdf:resource="http://th-owl/EC10-M04.owl#PhoenixDay1"/>
  <UGS:hasProcedure rdf:resource="http://th-owl/EC10-M04.owl#TIP1"/>
</owl:NamedIndividual>
<!-- http://th-owl/EC10-M04.owl#EIP1 -->
<owl:NamedIndividual rdf:about="http://th-owl/EC10-M04.owl#EIP1">
  <rdf:type rdf:resource="&UGS;Procedure"/>
  <UGS:hasNumber rdf:datatype="&xsd;string">1</UGS:hasNumber>
  <UGS:hasAction rdf:resource="http://th-owl/EC10-M04.owl#Action2"/>
  <UGS:hasAction rdf:resource="http://th-owl/EC10-M04.owl#Action3"/>
  <UGS:hasCondition rdf:resource="http://th-owl/EC10-
M04.owl#EIP1Condition"/>
</owl:NamedIndividual>
<!-- http://th-owl/EC10-M04.owl#EIP1Condition -->
<owl:NamedIndividual rdf:about="http://th-owl/EC10-M04.owl#EIP1Condition">
  <rdf:type rdf:resource="&UGS;Condition"/>
  <UGS:hasSensor rdf:resource="http://th-owl/EC10-M04.owl#CAST1"/>
  <UGS:hasTripType rdf:resource="http://th-owl/EC10-
M04.owl#EIP1SensorTripType"/>
</owl:NamedIndividual>
<!-- http://th-owl/EC10-M04.owl#EIP1SensorTripType -->
<owl:NamedIndividual rdf:about="http://th-owl/EC10-
M04.owl#EIP1SensorTripType">
  <rdf:type rdf:resource="&UGS;SensorTripTypeAny"/>
</owl:NamedIndividual>
```

SQL Triggers

- Store incoming events as rows in a database (one column for each possible field in the message)
- Create insert/update triggers in the database schema
- Triggers call external user defined functions to execute actions
- Excellent performance
- Poor extensibility (requires database schema changes)
- Query planners can result in unexpected behavior due to UDF side effects

```
SELECT CueOneCamera("Cam01",
   new.snsrLat, new.snsrLon,
   new.snsrAlt, 1),
   GrabVideoDelay(700,"Cam01")
WHERE new.eventID LIKE
   "TH.CAST1%"
    AND new.snsrLat NOT NULL
    AND new.snsrLon NOT NULL;
```

Scripting Languages

- Expose UGS system components to a scripting language
- Write scripts to represent mission programs
- Python
  - Currently used within the intelligence community
  - Interpreter (Jython) is too heavy, performs poorly on embedded platforms
- JavaScript
  - “Language of the web”
  - Interpreter (Mozilla Rhino) built in to the standard Java runtime
  - Fast performance
  - **Approach chosen for TerraHarvest**

JavaScript Mission Engine

- Rhino exposes Java APIs to JavaScript programs
  - Scripts can leverage any existing Terra Harvest functionality
- Terra Harvest framework (and the JRE) provides:
  - Asset (device) directory
  - Asset command and control
  - Posting and retrieving asset observations
  - Sending and receiving messages over custom communications channels (Java provides IP support)
- MissionProgramManager provides:
  - Persistent storage of missions
  - Execution of missions with runtime-configurable parameters
  - Predefined JavaScript variables for easy access to core framework services
  - Utility libraries for event handlers and threads

Example Mission


```
// trigAsset and camAsset are parameters set by the configuration GUI.
// trigAsset is the name of the asset that will be used as a trigger sensor.
// camAsset is the name of the camera to use to take a picture.
importPackage(org.osgi.service.event);
importPackage(Packages.mil.dod.th.core.observation.types)
importPackage(Packages.mil.dod.th.core.asset.types)
importPackage(Packages.mil.dod.th.core.asset.capability.commands)
takePictureObj =
{
    // Implement the Runnable interface, so this can be run as a thread.
    run: function () {
        // Get a reference to the camera object from the asset directory.
        // ads is a system-defined variable that points to the
        // AssetDirectoryService.
        cam = ads.getAssetByName(camAsset);

        // Tell the camera asset plugin to take the picture
        cam.captureData(true);
    }
}
```

Example Mission

```
eventObj =
{
    // Implement the OSGi EventHandler interface to handle events from the
    // observation store.
    handleEvent: function (event) {
        // Get the sensor report (observation) data from the event object.
        observation = event.getProperty("asset.observation");

        // Do some checks on the report to determine whether or not we want
        // to take a picture.
        if (observation.isSetDetection()) {
            sensings = observation.getDetection().getSensings();
            if (sensings != null &&
             sensings.get(0).getModalityType().equals(SensingModalityEnum.PIR)) {

                ls.info("Trip sensor has triggered the camera", []);
                // Start the background thread to take the picture.
                r = new java.lang.Runnable(takePictureObj);
                t = new java.lang.Thread(r);
                t.start();
            }
        }
    }
};
```

Example Mission


```
handler = new EventHandler(eventObj);

// Register the event handler with OSGi so we can get sensor reports.
// The system posts events to this topic when sensors produce data.
strTopic = "mil/dod/th/core/asset/Asset/MADE_OBSERVATION";
// Set an event filter so we only get reports from the asset we are
// interested in.
strFilter = "(obj.name=" + trigAsset + ")";

// ehh is a system-defined variable that points to the EventHandlerHelper,
// which scripts can use to easily register event handlers with OSGi.
ehh.registerHandler(handler, strTopic, strFilter); // registers handler with OSGi
```

Future Work

- Provide more helper functions and utility libraries to make initialization easier
- Add functions to stop running missions
- Replace the bundled Rhino (modified by Oracle) with the standard version from Mozilla
  - Allows scripts to extend Java classes
- Develop a mission library GUI
  - “App store” for predefined missions
  - User can easily select and customize missions when deploying sensors
  - Knowledgeable users/developers can create new missions and add them to the library

Questions?