KAoS Policy Development and Integration

Andrzej Uszok, Jeffrey M. Bradshaw
auszok@ihmc.us, jbradshaw@ihmc.us

Introduction

- The KAoS Services Framework can be straightforwardly integrated with existing software using the following steps.

Steps in Policy Development

- Setting up the general configuration
- Developing application-specific ontologies
- Making an ontology snapshot
- Configuring an ontology proxy
- Developing the Java ontology vocabulary
- Creating policies
- Saving a policy snapshot file
- Configuring the ant script to start the business logic code
- Integrating business logic code with KAoS
- Creating KAoS actors
- Deciding which policy services are needed
- Integrating policy services with business logic

Setting Up the General Configuration

- After installing KAoS in the selected location on the disk by unzipping the distribution file, the KAoS_HOME environment variable should be set to the root of the selected installation directory.
- KAoS requires:
  - Java 1.5 or higher
  - ant 1.7 or higher
- In other selected locations for integration development, the following directories should be created as placeholders for later configuration steps:
  - config
  - scripts
  - scr
  - lib

Developing Application-Specific Ontologies

- KAoS provides a set of generic ontology concepts needed for basic policy creation. These are available at: *http://ontology.ihmc.us/ontology.html*
- This set of concepts usually has to be extended with concepts specific to the application under development.
- The application ontology should contain definitions for all the concepts for which the business logic code can provide information, e.g.,
  - Through code instrumentation, the business logic for the control of radios can usually provide required information about transmission parameters.
  - Thus, all these concepts should be present explicitly in the ontology, in order that policies can refer to them.

Developing Application-Specific Ontologies (cont.)

- In general the following new ontology files are created:
  - Application**Action**.owl – containing definitions of application action classes with their specific properties
    - The new Action classes should be a subclasses of *http://ontology.ihmc.us/Action.owl#Action*
    - The properties of the Action classes should be subproperties of either *http://ontology.ihmc.us/Action.owl#hasDataContext* or *http://ontology.ihmc.us/Action.owl#hasObjectContext* in order to provide hints to KPAT about their importance
  - Application**Actor**.owl - containing definitions of application actor classes (or roles) with their properties
    - The new Action classes should be a subclasses of *http://ontology.ihmc.us/Actor.owl#Actor*
  - Application**Entity**.owl – containing definitions of application specific entities with their properties, which will be used to define context of the actions:
    - The new Entity classes should be a subclasses of *http://ontology.ihmc.us/Entity.owl#Entity (or more specific subclass)*

Making an Ontology Snapshot

- The ontology files should be configured with their **final static namespace** and should be loaded into the web server configured with a URL corresponding to this namespace
- Because access to the distributed web servers on which stored ontologies reside sometimes can be unreliable, slow, or inaccessible (e.g., no Internet connection, firewalls), KAoS provides a tool to create ontology snapshots.
- An ontology snapshot puts all the required ontologies in a single file.

Making an Ontology Snapshot (cont.)

- KAoS OntologyProxy can be started by calling target *ontology-proxy* from ant build *KAoS_HOME\scripts\kaos-tools*
- Going to the KPAT tab “Create Snapshot” allows the creation of a new snapshot:
  - A default KAoS ontology set should added to the snapshot:
    - Select from the pulldown list
    - Press the “Gather Ontologies” button at the bottom
  - Now, all the ontology files should be loaded into the snapshot
    - Paste their URL into the field
    - Press the “Gather Ontologies” button at the bottom.
- The tab “Edit Snapshot” allows modification or automatically refreshing of an existing snapshot. It is also very useful when ontologies have to be gathered from disparate domains (e.g. public Internet and corporate network).

Making an Ontology Snapshot (cont.)

- The ontology snapshot should be saved preferably into: *config/ontologySnapshots/SnapshotName.ont*

Configuring an Ontology Proxy

- The ontology proxy allows ontologies to be accessed locally, without the need for a network connection.
- In the *script/runProxyWithOntologySnap* create ant build file with target running OntologyProxy preloaded with this snapshot;
  - for example see target *ontology-proxy-for-flexfeed* in *KAoS_HOME/scripts/kaos-tools*

Developing the Java Ontology Vocabulary

- Development of code linking the business logic with KAoS policies and services requires references to the URLs of ontology concepts
- KAoS provides a simple tool to create Java constants for every concept defined in a given ontology with values equals to their URLs.
- In the *script/generateOntologyVocabulary,* create an ant build file with target running OntologyMapper for each defined ontology file
  - for example see target *vocabulary-selected* in *KAoS_HOME/scripts/kaos-tools*
- **No explicit URLs** should be used in the code.
  - Based on experience, such a practice creates difficult debuging problem (misspelled URLs, ontology changes).
  - Reruning the script automatically updates the URLs and concepts, and keeps the code consistent with the ontology.

Creating Policies

- Policies are created using KPAT, which currently provides two user interfaces: a traditional form-based GUI, and a hypertext GUI
  - Both interfaces provide similar capabilities
  - In both cases, the list of menu selections are dynamically obtained from the ontology repository based on the current context of policy creation
- A policy template definition capability allows the creation of simple custom interfaces to define a given kind of policy.

Saving a Policy Snapshot File

- Created policies and other concepts created in this process should be saved in policy snapshot file
  - This file can be created from KPAT tab “Configuration”
- The file should be saved in *config/policyConfigurationSnapshots/Name.cfg*
- In *scripts/runKAoSwithPolicyConfiguration* create ant build file with target running KAoS with this policy snapshot preloaded; make sure to set up the first two properties correctly

```
<property name="ontology.file" value="${basedir}/config/ontologySnapshots/ON.ont" />
<property name="directory.snapshot" value="${basedir}/config/policyConfigurationSnapshots/PS.cfg"/>

<target name="runKAoSwithL3078" description="Start KAoS with policy and ontolgy snapshot">
<fail unless="env.KAOS_HOME" message="Please set the environment variable KAOS_HOME" />

<echo message = "Starting ontology proxy, KAoS DS, Servlet and KPAT"/>
<parallel threadCount="4">
<ant inheritAll="true" antfile="scripts/kaos-tools/build.xml" target="ontology-proxy-autostart" dir="${env.KAOS_HOME}"/>
<sequential>
<sleep seconds="8" />
<ant inheritAll="true" antfile="scripts/kaos-core/build.xml" target="run-kaos" dir="${env.KAOS_HOME}"/>
</sequential>
</parallel>
</target>
```

Configuring the ant script to start the business logic code

- In order for existing applications to be integrated with KAoS at runtime certain parameters have to be provided. These parameters make KAoS functionality accessible from the modified application source code.
- The following elements are needed in the target starting the application:

```
<classpath>
    <!-- Include all jar files in the KAoS ${lib} directory -->
    <fileset dir="${env.KAOS_HOME}/lib">
            <include name="**/*.jar" />
            <exclude name="**/${optionalLib}/*.jar" />
    </fileset>
    <!-- Include the path to the config files -->
    <pathelement path="${env.KAOS_HOME}/${cfgPath}"/>
</classpath>

<jvmarg value="-Dkaos.core.service.boot=VMA.cfg" />
<jvmarg value="-
    Dkaos.core.policy.service.boot=${env.KAOS_HOME}/${config}/guardConfiguration.cfg" />
<jvmarg value="-Djava.util.logging.config.file=${config}/logging.properties" />
```

Integrating business logic code with KAoS

- If the jar libraries and other configuration parameters are correctly specified, then all available KAoS functionality can be accessed by retrieving the appropriate implementation of the given subset of functionality.
- This is done by calling a specific factory method on static class: *kaos.core.csi.CSIFactory*
- Such calls will create background connections with the KAoS Directory Service

Creating KAoS Actors

- The application must register itself as an actor or as multiple actors whose identities will be used to make policy decisions about its actions
  - Some class within the application code has to extend *kaos.core.csi.KAoSActorImpl* and call its *registerWithKAoS()*
  - This class can also register specific actor properties using addProperty()
    - e.g an actor representing a radio operator can have properties specifying his location, clearance, etc.

Policy Services available for business logic

- Inside the business logic the calls to KAoS Policy Service can inform the application about:
  - Authorization of an action – if an action is not authorized, an exception is thrown with information which policy decided it. (method *checkPermission*)
  - Additional obligations related to an action - a list of obligations for this actor are returned in a list sorted by the priority action description. In addition, if there are obligations for other actors, then KAoS will try to locate them and will send the appropriate actions to them. In order to accept such obligation actions, the actor has to implement a listener for KAoS CSI Requests. (method *getObligationsForTriggerCondition*)
  - Configuration options for the planned action – a range of allowed values for the given action properties is returned if a partial description of the action is sent to KAoS. (method *getAllowableValuesForActionProperty*)

Integrating Policy Services with Business Logic

- Action descriptions are the main datatype exchanged with KAoS:
  - kaos.core.csi.ActionInstanceDescription(Impl)
  - kaos.core.csi.OntPropertyDescription(Impl);
- Applications that check authorization policies must be able to create of action descriptions, and applications that handle obligations must be able to interpret them when received.
- Action descriptions can contain a complex value in the form of
  - kaos.core.csi.OntInstanceDescription(Impl);
- Names used as types and properties in these data structures are from the Java vocabulary files generated using the mapper.