Cougaar

Cougaar Agent Architecture Open-Source site

Frames Manual

Contents

- Overview
- Architecture
- Design
- Frameset XML
- Frameset DTD
- Code Generation

The Cougaar Frame Infrastructure: Overview

The Cougaar Frame Infrastructure provides a lightweight and efficiently distributed form of knowledge representation supporting two independent inheritance hierarchies, path following, and code generation. The specification of a given set of frame definitions (prototypes) includes not only conceptual information for whatever domain the frames are being used to represent, but also implementation hints to maximize efficiency. This allows the designer to fine-tune the definitions in a way that's not generally available in other frame systems.

Frames are Blackboard objects: new Frames and changes to Frames are published to a Blackboard associated with the FrameSet. Frames are also Java Beans, and can therefore be used by some rule engines (eg Jess, the Java Expert System Shell) in the pattern matching of the left-hand-side of a rule. Much of the power of the Cougaar Frame infrastructure comes from this unification between Frames in a knowedge base, entities on a Blackboard and facts in a rule engine.

For increased efficiency, Frames are organized into collectoins called FrameSets. FrameSets are used to cache relationships between Frames, in particular the containment relation. FrameSets also provide an implicit namespace of types.

The Cougaar Frame Infrastructure: Architecture

A *Frame* is a basic unit of knowledge, consisting of a collection of named *slots*, each of which can have a value. A *FrameSet* is an organized collection of Frames. The organization of any particular FrameSet is itself defined by a special subset of the Frames it contains. These metadata Frames are known as *PrototypeFrames*, while ordinary data in the FrameSet takes the form of *DataFrames*. Each DataFrame is an instance of a exactly one prototype, and can represent either a logical entity, or to binary relationships between entities. The importance of this distinction will become clear later.

PrototypeFrames define the slots that are included in DataFrames of the corresponding prototype (or of a descendent of that prototype). The definition of a slot has a number of dimensions, some of which are conceptual (eg the value type) and some of which are more like implementation pragmas (eg whether or not the slot should be represented as a data member in a Java class).

One somewhat unusual feature of slots is that their value can be specified as a *path* through the FrameSet. A Path is a first-class object in the specification of a FrameSet , and describes an ordered series of relative steps a given Frame should follow in order to find the current value of a path-valued slot.

The PrototypeFrames in any given FrameSet are arranged as single-inheritance hierarchy, much like the class hierarchy in Java. In addition, the Cougaar Frame System supports a second, independent hierarchy, known as *containment*, which for each FrameSet is defined a particular relation prototype. These two hierarchies can be thought of very loosely as “is a” and “is contained in”, though the doesn’t necessarily represent physical containment. Slots are fully inherited through the prototype hierarchy and inherited read-only through the containment hierarchy.

The Cougaar Frame Infrastructure: Design

A FrameSet in the Cougaar Frame Infrastructure is specified by a set of *PrototypeFrames*, which define the data types; and prototype inheritance specification; and a containment specification, which indicates which of the relation prototypes should define the containment hierarchy. A FrameSet , in other words, is defines a *domain* in the form of a dual hierarchy of types. It also implicitly defines a *namespace* of types. For now we support only single-inheritance along both hierarchies. The FrameSet has the responsiblity for handling per-prototype frame instantiation, as well as caching for efficient handling of relationship frames.

To facilitate instantiation, and also to increase the run time efficiency of slot lookups, a code generator generates a Java class for each PrototypeFrame. It is these classes that are instantiated for the *DataFrames*. Most generated classes extend another generated class: its parent in the prototype hierarchy. There are also two root prototypes in each domain, with no explicit parent: the root entity prototype, which is generated such that it directly extends DataFrame , and the root relation prototype, which extends the predefined class RelationFrame (in turn an extension of DataFrame).

The PrototypeFrame instances are kept at runtime in the FrameSet , along with the DataFrames. Although in principle all the necessary information could be generated in advance, thereby avoiding any need to represent Prototypes directly at runtime, in practice it’s useful to keep this meta-data available in the form of first-class Objects. On the other hand, PrototypeFrame instances are completely fixed at runtime: nothing about them can change. DataFrames are fixed only to the extent that their prototype can’t change. Otherwise they’re fully dynamic objects.

One of the novel features of the Cougaar frame system is that it supports two independent hierarchies. In addition to the prototype hierarchy, the Cougaar frame system a second single-inheritance hierarchy known as *containment*. Containment is a special relationship used to augment slot value lookup at runtime. As one might expect, DataFrames can include a frame-specific value for any given slot. If it does, no further work is required. If it doesn’t, the prototype hierarchy is searched for a default value or for a path specification (see below for more on paths). If this fails as well, the containment hierarchy can then be searched. This allows a DataFrame to have virtual access to slots defined by its logical container. The containment relation is defined as part of FrameSet and fixed when the FrameSet is created. This relation is an example of caching handled by the FrameSet.

Another novel features of the Cougaar frame system is the *Path* specification. The description of any given slot in a prototype can indicate that the value of that slot for some frame F is determined by following a particular path from F a runtime. A Path represents a kind of encapsulated visitor pattern. It consists of an ordered series of *Forks* and a slot. Each Fork refers to a relation and a role. This allows any given frame to follow a path, one relationship at a time, to any related frame, arbitrarily far away. When the final frame in the path is reached, the given slot value is returned as the value of the original slot reference. NB: Although DataFrames generally act like beans, including property-change support, Path-valued slots are an exception at the moment: if the value of Path-valued slot changes, listeners will not be notified. We hope to deal with this in a future release.

The Cougaar Frame Infrastructure: Frameset XML

FrameSets and Frames can be loaded into Cougaar via xml files. This section describes the XML elements and attributes for PrototypeFrames. The formal description is in framesets.dtd. The DTD format for the DataFrames of any particular domain is generated from the PrototypeFrames. Those are not described in detail here, but in general each Prototype has a corresponding element that can contains elements for the slots in that Prototype. Slot elements have only PCDATA for the value of the slot. For an example, see cougaar-topology.dtd and test-frames.xml.

A good choice for the doctype specification of this xml is as follows:

```
<!DOCTYPE frameset PUBLIC "-//W3C//DTD frameset" "http://www.cougaar.org/2006/frame
```

frameset

A frameset element is used to describe a FrameSet. It has four required attributes. The frame-inheritance describes the prototype inheritance structure. The only supported value right now is “single” (i.e., single inheritance). The container-relation attribute specifies which relation prototype defines the containment hierarchy. Any defined relation prototype can be specified here. The package attribute is used by the code generator: the classes generated for this frameset’s prototypes will be put in the given package and the files will be written to the corresponding directory stucture, following standard Java conventions. The optional index-slot attribute can be used to specify a primary index for fast reverse lookup (ie, finding a frame by slot value). The values of the given slot within the frames of any given prototype should be unique, and for now the slot should be declared immutable (the latter restriction will probably be relaxed in a later release). Providing an index slot will improve efficiency dramatically in large framesets particularly when the index is on the value slot of the frameset’s container relation. Finally, the domain attribute specifies a name for whatever logical domain the framset’s prototypes are describing. This name should be the root element in xml data files for this domain.

copyright

If a copyright element is present, its text is inserted at the top of of every generated java file for this FrameSet.

prototypes

A prototypes element can be used to group prototype elements. It has no attributes and is not required. Ordinarily a prototypes element would be within a frameset element in an XML file. But it can also be at the top level if the file is being read in to an existing FrameSet.

prototype

A prototype element defines a PrototypeFrame. It has one required attribute, name, which is the name of the prototype. It also supports three optional attributes: prototype, which specifies the “super” of this one; container, which specifies the prototype of frames that can act as containers for frames of this prototype; and doc, which will be used as javadoc in the generated class. The remaining structure of a prototype is specified by the slot elements it contains.

relation-prototype

A relation-prototype element defines a PrototypeFrame representing a relationship. The attributes are a superset of the attributes of prototype elements. The additional attributes, all optional, are parent-prototype, parent-slot, child-prototype and child-slot

path

A path element defines a Path. It has one required attribute, name. The elements it contains are any number of forks (order is significant) followed by an optional slot-reference.

fork

A fork element defines a “hop” in its enclosing path, where a hop consists of a relation prototype and a role the next object on the path should play in that relationshoip. Correspondingly, a element has two required attributes: relation (the relation prototype name) and role (“parent” or “child”).

aggregate-by

An aggregate-by element is used to specify that a slot value depends on an aggregate calculation applied to a set of related entities. The required attributes are relation (the name of the relation prototype) and aggregator (the name of a Java class that does the calculation). The aggregator attribute must name a class that implements org.cougaar.core.qos.frame.SlotAggregator. If the aggregator is not qualified with a package, it is assumed to be in the same package as the frameset.By default the frame is assumed to play the PARENT role in the relationship. This can be overridden with the role attribute.If any related entities are added or deleted, the aggregator is rerun automatically to compute a new value for the slot. Similarly, if the optional related-slot attribute is given and if that slot is modified in a related entity, the aggregator is rerun automatically on the related slot.

slot

A slot element is used within prototype element to define one of the prototype’s named fields. Slot elements have one required attribute, name, which is the name of the slot, and a range of optional attributes (see below). They may also have an optional *aggregate-by*subelement.

doc

This attribute provides optional documentation for the slot and is used as javadoc for the public accessor. It's included in the slot metadata.

immutable

This boolean valued attribute indicates whether or not the value of the slot in any given DataFramecan be chaged after initialization. The default is "false" (ie, values can be changed after initialization).

member

This boolean valued attribute indicates whether or not the generated code for this slot treats it as a data member or as a property. The former is more efficient for slots whose value is usually frame-specific, the latter is more efficient for slots whose value is usually defaulted. The default for this attribute is "true".Other slot attributes can render this one irrelevant. In particular, if a slot has a path, it will never be a data member, regardless of the setting of this attribute. Conversely, if the slot is declared to have a simple type, or if it's declared to be transient, it will always be a data member, regardless of the setting of this attribute.

notify-blackboard

This attribute specifies whether or not a change should be published on the Blackboard when the slot value changes. The default is "true".

notify-listeners

This attribute specifies whether or not PropertyChangelisteners should be notified when the slot value changes. The default is "true".

path

The presence of this attribute indicates that the default value for this slot is computed by following a path. The value of the attribute is the name of path. If a path attribute is present, the slot cannot be a member.

metric-path

The presence of this attribute indicates that the value comes from the Metric Service dynamically. The value of the attribute is a Metric Service path. If a metric-path attribute is present, it implies that the slot is read-only, member and transient, and has type "Metric".

inheritable-through

This attribute indicates whether or not the corresponding data member should be hidden from containment inheritance.

Set to “prototype-only” to inhibit containment inheritance. The default is “false”.

transient

This boolean valued attribute indicates whether or not the corresponding data member in the generated class will be declared transient. The default is “false”. Transient slots are always members.

type

This attribute is used to specify the Java type of values of this slot. The default is “String”. Other possible values are “int”, “long”, “float”, “double”, “boolean”, “Integer”, “Long”, “Float”, “Double”, “Boolean” and “Metric”. Slots with simple types must be members. See below for more on slot types.

units

This attribute provides optional units for the slot. It’s included in the slot metadata but currently isn’t used for anything else.

default-value

This attribute gives a default value of the slot. The value can be any string. If a slot has neither a value nor a path attribute, then the slot has no default value. In this case, the generated accessor code will issue a warning if it has no frame-specific value for this slot.

warn

This boolean valued attribute indicates whether or not a warning is generated at runtime if the slot has no value. The default for this attribute is “true”.

slot-reference

A slot-reference element can only be used within a path element, and if present, must be the last element in the path. It simply names a slot. As such it has one required attribute, name.

The Cougaar Frame Infrastructure: Frameset DTD

```
<!ELEMENT frameset (copyright?, prototypes)>
<!ATTLIST frameset frame-inheritance CDATA "single"
                   package CDATA #REQUIRED
                   container-relation CDATA #REQUIRED
                   domain CDATA #REQUIRED
                   domain CDATA #REQUIRED
                   index-slot CDATA #IMPLIED>
<!ELEMENT copyright (#PCDATA)>
<!ELEMENT prototypes (prototype | relation-prototype | path)*>
<!ELEMENT prototype (slot*)>
<!ATTLIST prototype name CDATA #REQUIRED
                    prototype CDATA #IMPLIED
                    container CDATA #IMPLIED
                    doc CDATA #IMPLIED
                    displaySlot CDATA #IMPLIED>
<!ELEMENT relation-prototype (slot*)>
<!ATTLIST relation-prototype name CDATA #REQUIRED
                             prototype CDATA #IMPLIED
                             container CDATA #IMPLIED
                             parent-prototype CDATA #IMPLIED
                             parent-slot CDATA #IMPLIED
                             child-prototype CDATA #IMPLIED
                             child-slot CDATA #IMPLIED
                             doc CDATA #IMPLIED>
<!ELEMENT path (fork*, slot-reference?)>
<!ATTLIST path name CDATA #REQUIRED>
<!ELEMENT slot (aggregate-by)?>
<!ATTLIST slot name CDATA #REQUIRED
               type CDATA #IMPLIED
               default-value CDATA #IMPLIED
               units CDATA #IMPLIED
               path CDATA #IMPLIED
               metric-path CDATA #IMPLIED
               doc CDATA #IMPLIED
               member (true | false) "true"
               warn (true | false) "true"
               immutable (true | false) "false"
               notify-blackboard (true | false) "true"
               notify-listeners (true | false) "true"
               inheritable-through (all | prototype-only) "all"
               transient (true | false) "false">
<!ELEMENT fork EMPTY>
<!ATTLIST fork relation CDATA #REQUIRED
               role CDATA #REQUIRED>
<!ELEMENT slot-reference EMPTY>
<!ATTLIST slot-reference name CDATA #REQUIRED>
<!ELEMENT aggregate-by EMPTY>
<!ATTLIST aggregate-by relation CDATA #REQUIRED
          role (parent | PARENT | child | CHILD) "parent"
          aggregator CDATA #REQUIRED
          related-slot CDATA #IMPLIED>
```

Code generation for the Frame Infrastructure is handle by the class FrameGen. It reads an xml file in frameset.dtd format and generates a Java class for each prototype. In addition it generates a dtd for the domain. It can also generate a uml file (.xmi) for the domain, as well as a Jess file (.clp) for the shadow classes if you want to write Jess rules that operate on frames.

Several features of frameset.dtd are specifically there for the code generator, generally for efficiency but in some cases simply as useful information. In the former category are most of the attributes on a slot element: type, default-value, member, immutable, notify-blackboard, notify-listeners, transient. Semantic details can be found the Frameset XML section. In the latter category are the package and domain attributes of the frameset entity.

The code generator could be extended to generate other things as well, including domain-specific runtime parsers for data frame xml files and domain-specific FrameSet classes.