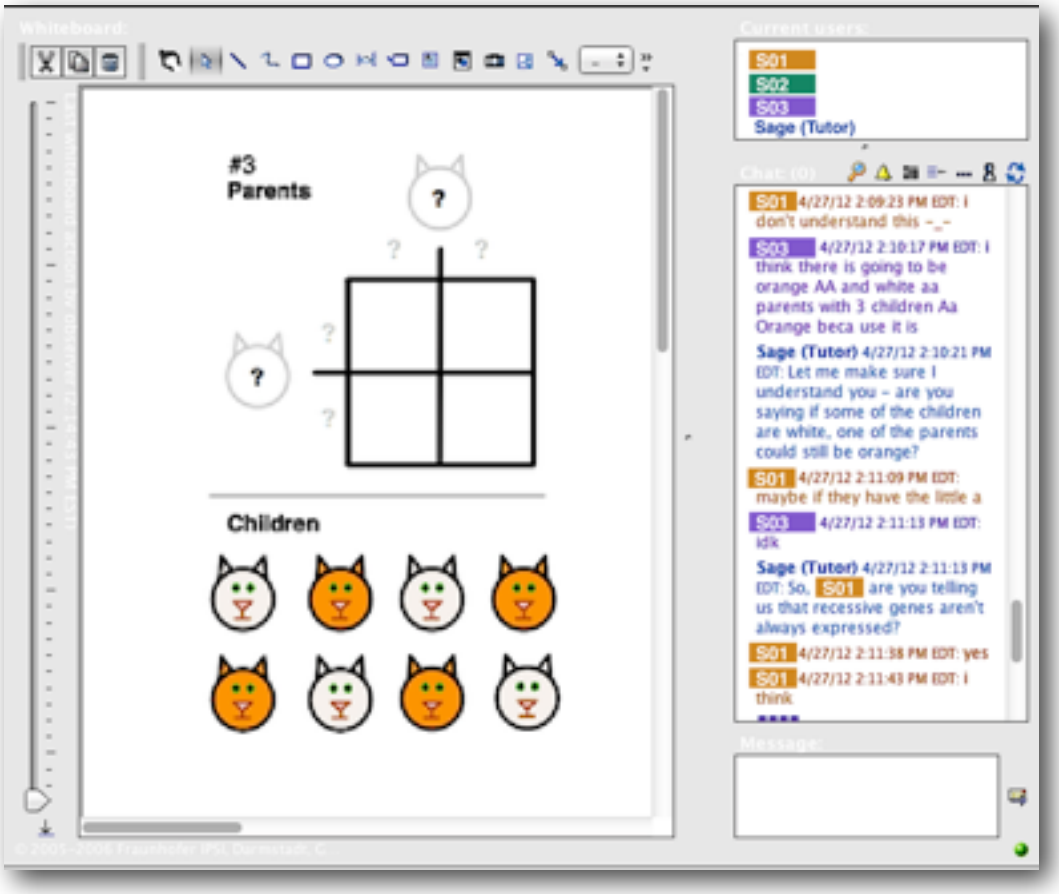
Bazaar!

an architecture for conversational agents

David Adamson, Language Technologies Institute
dadamson@cs.cmu.edu



Bazaar Agent Architecture Overview

Getting Started

0. The Bazaar Distribution

Download Bazaar from http://cs.cmu.edu/~dadamson/bazaar/Bazaar.zip and extract the folder somewhere appropriate.

This contains the core Bazaar libraries and an example agent, plus this document.

1. External Dependencies

Eclipse will be our development environment. http://www.eclipse.org/downloads

ConcertChat is a collaborative chat and whiteboard environment that we'll use to interact with our agents. Other user-facing ends can be used, with appropriate adapter classes.
http://cs.cmu.edu/~dadamson/bazaar/ConcertChatServer.zip

MySQL – only required if you're running your own ConcertChatServer (step 3)
Download MySQL from http://dev.mysql.com/downloads/mysql/5.0.html Use the appropriate version for your platform, and follow the standard installation instructions.

2. Open the Projects

Import the three projects (BasilicaCore, BaseAgent, and RepeatExampleAgent) into Eclipse.

3. Set up a chat environment

If you're using an existing ConcertChat server or another chat environment, ignore this step.

Make sure MySQL is running, then launch the ConcertChat server:
cd ConcertChatServer/bin; ./start.sh (Mac OS X and Linux)
(see additional setup instructions in ConcertChatServer/README.txt)
Also create a chat room instance from the ConcertChat administration webpage:
http://localhost:9000/concertChat/admin

4. Launch your agent

Run **RepeatAgentOperation.launch** from the **RepeatExampleAgent** project in Eclipse and verify that you can join a chatroom interact with the example agent, using the room you created in Step 3 (or in an existing room on whatever chat server you're using)
(this just runs myagent.operation.RepeatAgentOperation, specifying the "runtime" folder as the working directory)

Also launch a chat room client from the ConcertChat administration webpage.

About Your Agent

Configuration and Runtime Directory

- For a cleaner root directory, **runtime** is specified as the working directory for each agent project. This can be set in Eclipse->Run->Run Configurations->Arguments->Working Directory. For the provided agent, this has already been set in the given **.launch** file.

- The **runtime/properties** folder contains Java properties files for individual agent components. Most of them follow the naming convention "ClassName.properties"

- **system.properties** includes configuration information for the chat environment.
  If you host the ConcertChat server somewhere besides localhost, change the properties *agilo.client.properties*, *agilo.logging.log4jConfig, agilo.server.name, agilo.webserver.name,* and *ipsi.concertchat.menuconf* to reflect this.

- **operation.properties** specifies which components are in use for this agent.
  Preprocessors handle events sequentially, in the order you list them in.

- **agent.xml** is where the agent's name is set, and where you'd hook in another environment besides ConcertChat.

- Other folders are referred to by various components (for example, the *MessageAnnotator* uses the files in the **dictionaries** folder, and the *PlanExecutor* refers to the **plans** folder.) - most of these locations are configurable from the component's properties files.

Defining And Responding To Events

- An **Event** is a simple object representing something interesting that's happened in the chat room, and contain the necessary information for a receiving module to respond to them. There are raw Events like **MessageEvent** and **PresenceEvent,** representing the actions of chat---room participants, and Events that result from analysis of raw events or that represent system state, like **DormantStudentEvent** or **LaunchEvent.**

- A BasilicaListener or BasilicaPreProcessor registers itself to respond to the classes of Events returned by its ***getListenerEventClasses()*** or ***getPreProcessorEventClasses()*** methods. Only events of these classes will be passed to the component.

- Your components can create and listen for existing classes of Events, or you can define new subclasses that are particular to what's "interesting" to your agent. In either case, keep in mind that other components may respond to or create the same events that yours do – design your components and your events accordingly.

Bazaar Agent Architecture Overview

Defining Listener and Preprocessor Components

In **operation.properties,** specify which PreProcessor and Listener components should be loaded for the agent, by their full classnames. These handle analysis of and response to events, respectively, and implement the interfaces described below.

BasilicaPreProcessor: *preprocessEvent(InputCoordinator source, Event e)*
The "first wave" of analysis and event generation. Things you can do in response to an event:

- Modify the given Event (add annotations, etc).
  It will be passed forward to the second-stage listeners.

- ***source.addPreprocessedEvent(e)*** add a new event to the second-stage "reaction" queue.

- ***source.queueNewEvent(e)*** Queue a brand new event to pass both waves of processing.

BasilicaListener: *processEvent(InputCoordinator source, Event e)*
The "second wave" of event responders. Things you can do in response to an event:

- ***source.addEventProposal(Event)*** Propose an agent action in response to this event

- ***source.addProposal(PriorityEvent)*** Propose an agent action in response to this event, with a specific proposal behavior (block other events, callback on accept, etc).

- ***source.pushProposal(PriorityEvent)*** Push a proposal directly onto the output queue - this is preferred when behaving asynchronously (in response to a timer, etc)

BasilicaAdapter implements both of these interfaces and provides other conveniences, for example loading .properties files into a local dictionary.

Proposing Actions

- A BasilicaListener can propose new actions to be taken in the chat environment by passing them to the output queue (by way of the InputCoordinator's ***add/push Proposal*** methods). Note that the default OutputCoordinator only knows how to enact MessageEvents and WhiteboardEvents.

- Proposals (called **PriorityEvents)** are constructed with a priority value and a timeout in seconds – other events in the output queue will be considered (and eventually accepted or rejected) based on these values.

- Each PriorityEvent is also associated with a **PrioritySource** –once a proposal has been accepted, its source will have influence on which subsequent proposals can be accepted.

- Convenience methods for creating proposals with certain kinds of priority sources are offered by the **InputCoordinator** and **PriorityEvent** classes - see the usages in RepeatExampleAgent, and the comments in those classes.

Bazaar Agent Architecture Overview

A Bazaar Glossary:

Bazaar is evolving, and several components of the codebase have old names that are no longer fully descriptive of their function. To aid in your translation of theory to implementation:

Event == Event / Action / Trigger

everything passed from one component to another
including input events from the environment, events created by preprocessors,
and PriorityEvents (which wrap regular MessageEvents/WhiteboardEvents)

PriorityEvent == Proposal / Action / Output Event

includes static factory methods to create special kinds of proposals
defines callbacks to notify components of proposal acceptance / rejection

PrioritySource == Proposal Source / Lingering Advisor

defines proposal priority logic & blocking-advisor functionality.
subclasses implement common cases (used by PriorityEvent factory methods)

BasilicaPreProcessor == Watcher / Listener / Detector / Analyzer / Component

first set of components to process/analyze incoming events.
specified in *operation.properties*
PreProcessors are run sequentially in listed order.

BasilicaListener == Reactor / Actor / Component

second set of components to process/respond to pre-processed events.
specified in *operation.properties*
Reactors are run sequentially in listed order.

BasilicaAdapter == BasilicaPreProcessor+ BasilicaListener

implements common+convenience methods for both component interfaces
recommended superclass for most Bazaar components

InputCoordinator == Event Source / Pipeline / Event Coordinator

Receives & relays events to each stage of the Bazaar pipeline
InputCoordinator also manages access to the OutputCoordinator queue.
When adding proposals, use *addProposal* methods if reacting to a "fresh" event.
use *pushProposal* methods otherwise (i.e., in response to a timer).

OutputCoordinator == Proposal Queue

Manages action proposals as relayed by the InputCoordinator
Updates proposal priority with advice from recent actions' PrioritySources
Agent authors should not need to interact directly with the OutputCoordinator.
Accepts and rejects events based on priority/timeouts
(and invokes accept/reject callbacks)
installs and manages PrioritySource advisors.

Bazaar Agent Architecture Overview

Timers, Plans, and Other Advanced Topics

- Many components implement the **TimeoutReceiver** interface (or create an anonymous inner class of one), allowing behaviors to be triggered by a **Timer's *timedOut*** method.
  Use ***source.pushProposal*** to queue action proposals in response to a timeout.

- Scripts using the **PlanExecutor** component deserves its own tutorial. However, the documentation in **basilica2.agents.listeners.plan** should be sufficient, if spread out.
  The RepeatExampleAgent also executes a simple timed plan.

- **TuTalk** http://www.pitt.edu/~tutalk/ may be used within Bazaar to power hierarchical dialogues – see the **TutorAgent** example at http://cs.cmu.edu/~dadamson/bazaar/TutorAgent.zip

- Models trained in **LightSIDE** http://cs.cmu.edu/~emayfiel/side.html may be used by Bazaar to classify text – see the **AnnotatorExampleAgent** example at
  http://cs.cmu.edu/~dadamson/bazaar/AnnotatorExampleAgent.zip
  You'll have to import LightSIDE as an Eclipse project as well.

- Recorded chatroom transcripts can be played back with the **ChatterBox** agent at
  http://cs.cmu.edu/~dadamson/ChatterBox.zip

Questions? Contact dadamson@cs.cmu.edu