Chapter 9
Collaborative Systems

What is a Collaborative System ?

A system where multiple users or agents engage in a shared activity, usually from remote locations.

Characteristics of CS

- agents working together towards a common goal
- have a critical need to interact closely w/each other
  - sharing info
  - exchanging requests
  - checking on each others' status
- have certain level of concurrency
  - agents interacting with system and with each other at roughly the same time

Examples

- A chat session is a CS
- An e-mail client is not

Elements of a CS (or DS)

- Autonomous or user-driven *agents*
- Operational or data *servers*
- Dynamic and persistent *data repositories*
- *Transactions* between agents, servers, and data

CS Examples

- Shared whiteboards
- Interactive chat

Autonomous:

- Distributed or parallel compute engines
- Coordinated data search agents (e.g., web robots)

Issues with COllaboration

- Communication needs
- Maintaining agent identities
- Shared state information
- Performance

Communication needs

- Must be flexible in its ability to route transactions
  - underlying communication must support
    - pt-to-pt messages between agents,
    - broadcast messages
    - "narrowcast" or multicast messages
    - passing objects
- E.g., An interactive chat server supporting chat rooms

Maintaining agent identities

- To be able to address and deliver messages,
- To restrict access to certain resources by using authentication
- To maintain resources associated with individual agents

E.g.,shared whiteboard application: a virtual drawing space that multiple remote users can view and write on in order to share information, ideas, etc.

Shared state information

- Data and resources are shared among participants in CS
  - A cooperative effort among computing agents is usually expressed in terms of a set of shared data

E.g., in shared whiteboard application, a reasonable and consistent way to determine how to merge incoming requests.

Performance

- There exists a tradeoff between keeping shared state consistent across all agents and maximizing the overall performance.
  - A central mediator acting as a clearinghouse for agent events
    - updates are sequenced correctly across all agents
    - can become a bottleneck
  - Peer-to-peer system
    + better performance
    - difficult to maintain concurrency

A Basic Collaborative Infrastructure

- Involves a single mediator (the server) handling interactions among multiple collaborators (clients).
- Each collaborator has a unique identity, issued by the mediator
- Each collaborator can
  - either broadcast messages to all of the collaborators registered with the mediator,
  - or it can send a message to a single collaborator.

The right communication scheme ?

- basic socket communications
- ⇨ message passing
- ⇨ RMI remote objects
- CORBA remote objects

Building the Infrastructure with Message Passing

- We start with message-passing framework of Ex 6-10, Ex 6-11.
- Multiple agents pass Messages to each other through a single MessageHandler
  - Each Message object has an identifier and a set of arguments.

MessageHandler class

- read messages from network
- construct Message objects from the data received
  - according to message identifier, choose the right Message prototype from the list
  - The set of Message prototypes serves to define the message protocol that the MessageHandler understands and can be updated on the fly if needed
- call the Message's Do() method to handle the message locally

MessageHandler of Ex 6-10 is not sufficient

- Only supports point-to-point message passing
- Options:
  - create a MessageHandler object for each agent we want to talk to
  - upgrade MessageHandler to manage multiple network connections to agents

Multi-Agent Message Handler Class

Two utility classes:

- AgentConnection
  - holds a pair of input and output streams connected to a remote agent
- AgentHandler
  - takes care of listening to a particular agent for asynchronous messages

Updated MessageHandler class

Ex 9-1

- MessageHandler maintains a table of agent connections, associating each connection with an ID number (see buildMessage method)
- A set of methods for adding, removing, and getting agent connections has been added:
  - addAgent()
  - removeAgent()
  - protected getAgent() – to get the associated AgentConnection
- To be able to specify which agent to talk to readMsg() and sendMsg() were overridden

Adding a new agent to MessageHandler

- one of the addAgent() methods is used
- an AgentConnection is made
  - to hold InputStream and OutputStream connected to the agent
  - the connection is stored in a Hashtable using the agent's ID number as the key, along with a reference to the MessageHandler
  - a new thread is started for AgentHandler

- Synchronizations are necessary
  - in readMsg() and sendMsg() to synchronize on the input and output streams of each agent.
  - in all methods for adding and removing agents from the MessageHandler to allow asynchronous agent handling.

Building Collaborative Infrastructure

- Collaborators – agents that work together towards the common goal of the system
- Mediators – serve to facilitate the communications among the collaborators

The Identity class

- CS need to provide an identity for each agent in the system,
  - so that transactions can be targeted and traced to individual agents
  - Properties list
    - name property – a descriptive name
    - id property – an internal identifier used to tag each collaborator uniquely
    - further properties in the form of serializable objects
  - Why implements Serializable interface?

A collaborator needs to …

- have a unique identifier in the system, so that messages can be routed to it
- be able to connect to mediators, or to other collaborators, to engage in communication with them
- be able to connect to send messages and to be notified of incoming messages

A Collaborator Interface

Example 9-3

- getIdentity() – returns an identity
- connect() – opens a remote connection to a remote mediator or collaborator
  - arguments specify how to locate the agent on the network
- send() and broadcast() to send messages
- notify() – through which messages are received

A mediator needs to

- be able to register new collaborators by providing them with unique identifiers
- send messages to individual collaborators
- broadcast messages to all collaborators that it has registered

A Mediator Interface

Example 9-4

- newMember() – generates a unique id for a new collaborator
- removeMember() – removes collaborator from Mediator's registry
- send() and broadcast()

Same interfaces can be used whatever the underlying communication scheme is.

MessageMediator class

- implementation of Mediator interface
- uses MessageHandler to route messages back and forth between remote agents
- Has a
  - a MessageHandler to route messages
  - a ServerSocket to accept socket connections from remote agents
  - a port number that it listens to for connections
- It also implements Runnable interface

run() method

- creates a ServerSocket listening for asynchronous connections from agents at its designated port - loops continuously trying to accept connections over the socket
- When a new connection is made, a new agent is added to the handler by calling its addAgent() method
- Mediator: creates a new Identity for the agent and sends a message to the agent containing its identity

A Mediator Based on Message Passing

Example 9-5