# Distributed Collaboration - Assignment 3: Adaptive Distributed Architecture

**Date (formally) Assigned: October 10, 2012**

**Completion Date: (~~Monday~~Friday) October 26~~2~~, 2012**

Objectives:

- Implement P2P communication
- Separate session management from relaying.
- Separate distributed functions from non-distributed functions
- Support dynamic adaptations

So far all of you have a central server that provides clients two services:

- Session management: a client can discover which other clients exist.
- Relaying: a client can send messages to other clients through the server.

These two services are probably bundled together in your implementation. You should now separate them so that clients can also communicate messages with each other directly without communicating through the server. They will still use the central session management to discover each other. After this discovery, they should be able to run in one of two modes:

Relay mode: as before, they send messages through the server.

P2P mode: they can send messages to each other directly.

You can assume that each clients either uses the Relay mode or P2P mode for sending messages out, and for extra credit you can allow a client to send relayed messages to some clients and direct messages to others. For extra credit, you can allow clients to synchronize their mode changes. You can use ObjectEditor to interactively change the mode change or use custom widgets.

If you implemented latecomer support in the central server, you can assume that this facility is available only if some clients is the relay mode, though for extra credit you can provided when all messages send messages directly to each other.

To support the P2P mode you will need to extend the session management service to not only allow clients to know when other clients arrive and leave but also to fetch and register remote objects that allow clients to talk to each other directly.

Ideally, the P2P and relay mode should be implemented by separate (sets of) classes which are distinct from your model classes. One way to do this is to define an out coupler and in coupler object in each client. The out coupler is an observer of the local model and sends model changes to remote models. The in coupler is an object that receives model changes and calls corresponding write methods in the model.  The information sent out by the out coupler to a remote in coupler can be different from the information it receives from the model.

Ideally, the out coupler should send the information through either a P2P or relay sender, implement a common interface used by the out coupler.  Similarly, the in coupler does not receive information directly from the remote computer. Instead it receives this information from a P2P or relayer receiver. This will make the models and couplers independent of not only the mode but also the underlying communication mechanism.

You should demo the communication paths using the Tracer facility described in class. In addition, you should define a TracerBus listener of your own that logs all message sent and received by a client/server and displays this log in either an ObjectWIndow or your own implementation. You can see the YouTube fault tolerance video to see illustrate event waiting.

Add delays to the communication links to show that causality consistency problems arise in p2p communication but not relayed communication, and  p2p communication does not have an extra hop.  As my demos showed, causality issues arise when a message arrives at a site before its causal message(s) that, that is, messages that caused that message to be sent.

## Submission

Create and post a video showing the mode changes, message paths, delays,  logs and other interesting aspects of your implementation. In addition, give me a printed document giving

1. The interfaces of the remote objects registered by the clients and server.
2. The common interface implemented by the p2p and relayer sender to get messages from the model (through ideally an out coupler).
3. The interface used by the relayer and p2p receivers to deliver message to the model (through ideally an in coupler).
4. A~~a~~ll the regular credit features you have implemented
5. A~~a~~ll features you have implemented that go beyond what was asked. These could include the extra credit features I suggested and additional features you chose.
6. R~~r~~easons for supporting both the P2P and relaying mode, dynamically changing it, and allowing each sender, receiver pair to decide the mode used to communicate messages from the sender to the receiver.