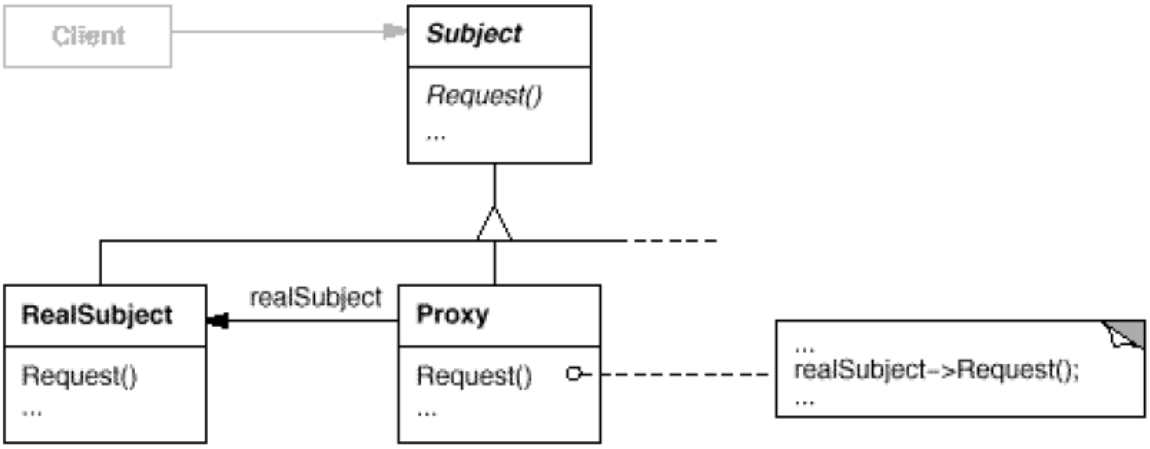
Name: Kevin DeVocht
CS 340 Fall 2014
Woodfield's Section

Design Principles & Patterns

- Proxy Pattern

The Proxy pattern is in a sense an interface. Like an interface, the code behind the proxy can be modified without affecting code that is interacting with the proxy. A proxy allows more in depth control to the object behind the proxy when compared to an interface. This is the main reason to use it over an interface. For example, we did not write the server for Phase 1, but because we created a server proxy that interacts with the server, we can create our own protocols on the client side that make the most sense to us and make our lives easier. In a later phase, we will write our own server, If we want to change anything on the server, the only class we will need to change is the server proxy. On page 234 of Design Patterns it states that "A remote proxy provides a local representative for an object in a different address space." This is exactly the case that we have with our project. This is why we have implemented the server proxy class which, like it's name implies, is a proxy. It allows us to interact with objects that are normally remote. We can also create mock servers for testing purposes and any code interacting with the server proxy has no idea nor cares which server the server proxy is communicating with. So long as the proper protocols are followed.

UML from Design Patterns page 235:



The source coding showing the proxy:

```
	public ClientCommunicator(IServer server) {
		this.setServer(server);
		this.setPoller(ServerPoller.getPoller(this));

	}
```

In the above function we see that ClientCommunicator takes an IServer as a parameter and the Subject corresponds with subject in the UML.

```
	public class ServerProxy implements IServer
```

Name: Kevin DeVocht
CS 340 Fall 2014
Woodfield's Section

We the see from the class declaration of the ServerProxy that it implements IServer. So ClientCommunicator can use a ServerProxy. The ServerProxy is the Proxy in the UML.

```
	public <T extends IResponse> T sendCommand(IRequest request, T emptyResponse)
			throws Exception {
		BufferedReader response = this.getResponse(request);

		return (T) emptyResponse.fromResponse(response,
				cookieManager.getCookieStore());
	}
```

This is the sendCommand method that correlates with the Request method on the uml

```
	public boolean registerUser(String username, String password) throws Exception {
		RegisterUserRequest request = new RegisterUserRequest(username, password);
		FailSuccessResponse response = new FailSuccessResponse();

		try {
			return server.sendCommand(request, response).isSuccess();
		} catch (RequestException r) {
			return false;
		}
	}
```

here we see that the registerUser method on the ClientCommunicator is just using the sendCommand method again correlating with the Request method on the UML.

```
	public <T extends IResponse> T sendCommand(IRequest request, T emptyResponse) {

		return (T)request.getDefaultResponse();
	}
```

This is the sendCommand method from the MockServer which correlates with the Request method on the RealSubject of the UML.

So as we can see from the code, we can easily swap out the Mock server for a real server and the only thing we have to do is change the server that the ServerProxy is pointing to.

- Data Integrity Enforcement in the Model
  The only Data Integrity Enforcement in Phase one is through our canDo methods. The way that we enforce data integrity is to call one of the canDo methods before calling the server command. An example of this would be the canPlaceCityAtLocation method:

```
	public boolean canPlaceCityAtLocation(VertexLocation location, int playerIndex,
String status){
```

Name: Kevin DeVocht
CS 340 Fall 2014
Woodfield's Section

```
		if(!status.equals("playing"))
			return false;
		return isSettlementIsOnLocation(location, settlements, playerIndex);
	}

	private boolean isSettlementIsOnLocation(VertexLocation location, List<ICommunity>
list, int playerIndex) {
		for(ICommunity settlement : settlements) {

if(location.getNormalizedLocation().equals(settlement.getLocation().getNormalizedLocation
())) {
				if(settlement.getOwner().getPlayerIndex() == playerIndex)
					return true;
			}
		}
		return false;
	}
```

This method makes sure that the vertex the player is trying to place a city on has a settlement already and that the player owns the settlement. So if this function is called before sending a request to the server to place a city at a specific vertex, we can ensure that the preconditions of the server(valid vertex location) is met. If all canDo methods are implemented properly, we canl guarantee that that the server will return a valid model because we met the preconditions of a valid model for the server.

Team Report

Time Spent Implementing Phase 1:

Name	**Scale**	**Comments**
Curtis Wigington	5+	He was the MVP of this Phase. Without him we would not have completed the project on time and we would not have had as solid of test cases as we had
Jacob Glad	3	Was in charge of Server proxy and did most of it on his own. The reason why I did not score him higher was because he disappeared for Wednesday afternoon through Thursday. We as a group, we did not know what he had completed until Friday afternoon.

Name: Kevin DeVocht
CS 340 Fall 2014
Woodfield's Section

James Hutchings	2	Not sure what is going on with him. He has missed a lot of class and therefore team meetings after class. I am totally fine if things come up as long as it is communicated to the team. He had none of the poller done before arriving on Friday and left before completing all test cases, to go to the football game.
Kevin DeVocht	3	I mainly worked on the map. The reason why I did not score myself higher was because after implementing all the logic for the map, I had to have Curtis sit down with me and refactor it and fix several bugs.
Mitchell Tenney	4	Was always asking what he could do to help out, finished all the little details before submitting the project for the team. Went in on Monday morning to pass off the test cases with the TA's. He was an impromptu Project Manager and did a great job.

Scale : 5 = Excellent, 4 = Very Good, 3 = Good, 2 = Unsatisfactory, 1 = Disaster

Over all I felt our team worked together really well. The biggest problem we encountered was lack of communication. As the deadline was approaching both James and Jacob stopped responding to our attempts to communicate to me. To me the day before a project is due is a crucial time to be in constant communication with your team. Even if you can't work on the project, thats fine, just let us know what you did get done and when we can expect to see you back. I realize this was the first phase and we are working out the kinks on our team. I am not worried about the next phase and feel like we can do well moving forward.