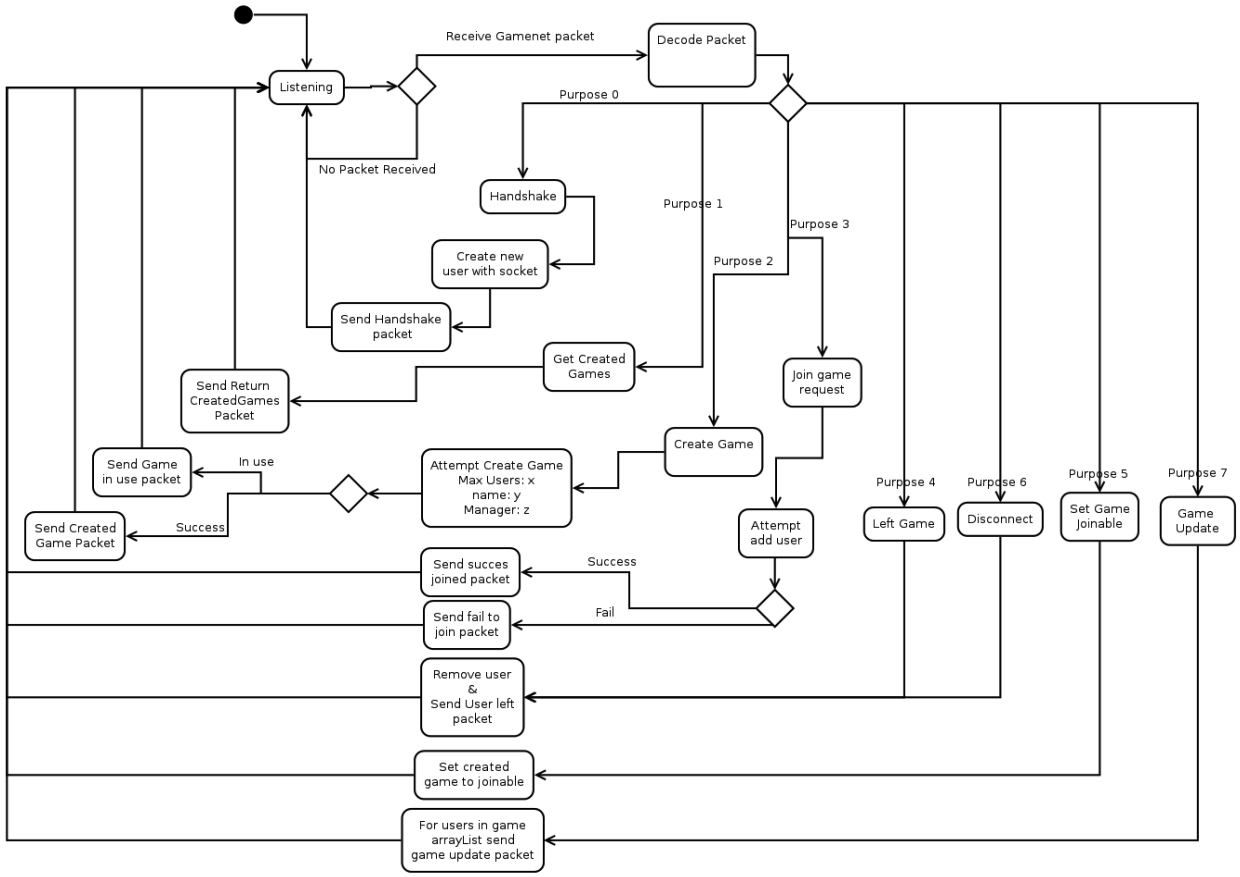
Marc Touchette #70489980
Andrew Beier #74688147
Armando Pensado Valle #34043453
Informatics 221 - Winter 2013

Project Parts 1 and 2

Game Description

Our version of the Lunar Lander game takes a twist on the idea of safely landing on the moon. In our version the objective of the players is to race between the moon and their Orbiter to build their moon colony the fastest. The players will have to dock with their Orbiter ship to receive base parts and land on the moon with a slow enough speed to prevent crashing in order to release their payload. The first player to build their complete base will win the game. The overall architecture of our system is an event based network with object oriented software on the clients.

GameNet Server UML Activity Diagram



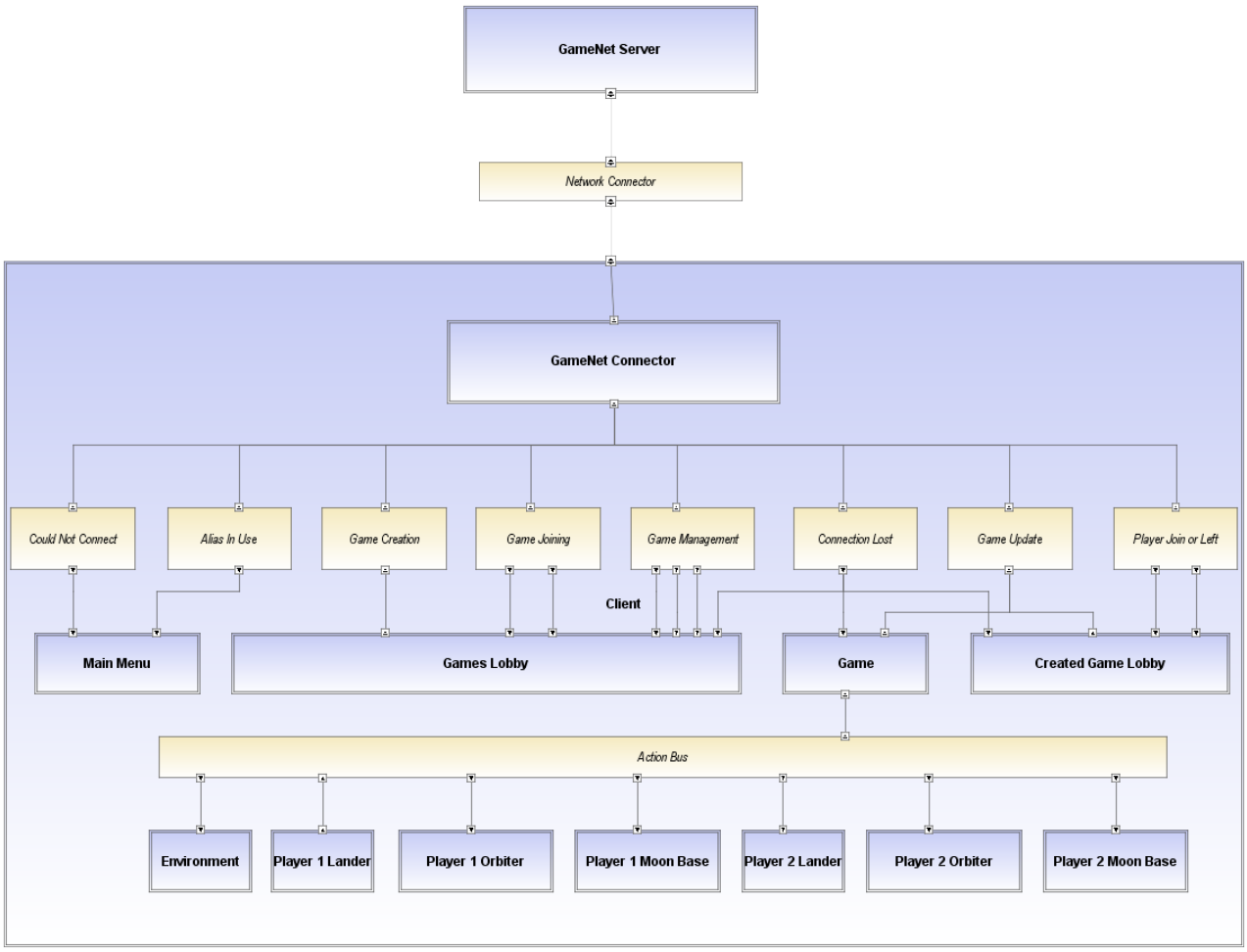
When we began designing our lunar lander game we knew that we needed some form of networking. Originally, we discussed a direct connection, however, we ultimately decided to use a piece of software a member of the group had worked with before. A former student at UCI, Jonathan Farnworth, wrote the GameNet software. As none of us had directly worked with the software, it was important to understand its architecture at least at the interface level to use it as our event bus.

We chose to use a UML Activity Diagram to model GameNet, because the source code was simple enough that it could be completely read and analyzed. We also knew we needed a model that thoroughly explained the interface of the software. This was an excellent exercise in architecture recovery. The nature of the system lent itself well to an Activity Diagram because it is threaded and each thread is essentially a listener that passes events, as well as managing a small database of active games and users. Even with a small piece of code, only seven classes and one to two thousand lines of code, analysis of the architecture was not easy. All of the classes needed to be read and understood which took a lot of time. Ultimately, the model that we produced is a great representation of how the packet passing is handled by GameNet and helped us to decide on the major architectural decisions for the rest of our game.

Using UML worked well for the purpose of capturing the architecture of GameNet. It is widely used so the tools are well developed and easy to use. It did not, however, allow us to specify hardware aspects of GameNet. In this case, the simplicity of the model was actually more desirable. We were able to focus entirely on the software and how it would factor into the architecture of our Lunar Lander game. We also discussed making the model more visually appealing. We decided, however, that the visualization of the model was designed to show us how our event bus would interact in our system and that it had served it's utility as it is shown.

Working with modeling tools like UML to design a system before implementing it greatly improves efficiency, coordination, product quality, and communication between developers. This case is no exception; all of us are able to quickly reference the models we have developed to see how the system is supposed to interact between the components. This will allow us to work in parallel and not trip over each other as we all have a single vision that has been modeled and well understood. This is especially true when using third party software. Rather than coding and trying to figure out the API as different problems are encountered, using a model like the Activity Diagram of GameNet allows us to use the resource efficiently even without any documentation of the software.

System xADL Model

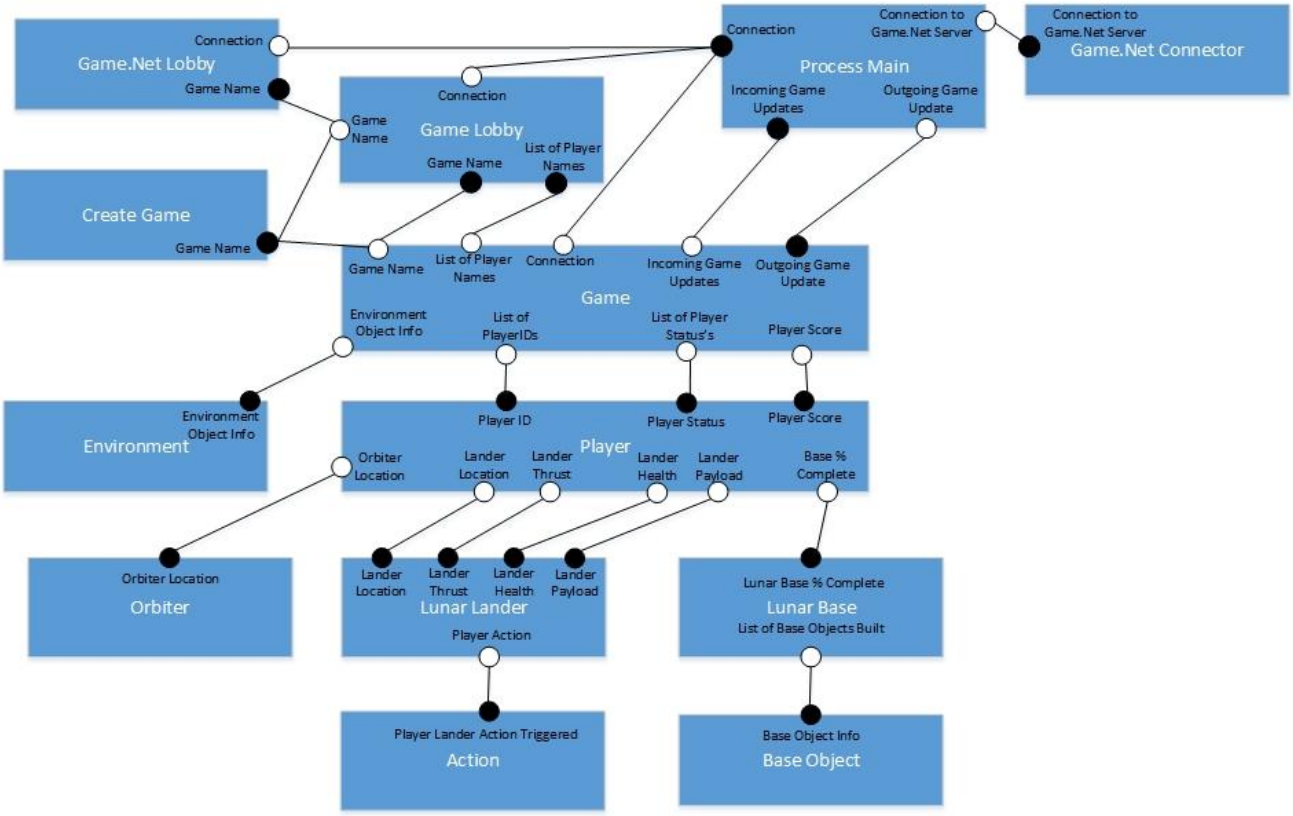


The xADL notation turned out to be an excellent choice for modeling the structure of the overall system from an event-flow point of view. The xADL way of modeling, which treats connectors as first-class entities, forced us to think carefully about how the components of the game should communicate. For example, the client communicates with the server via the GameNet connector. However, despite its name, we realized it is better represented as a component that communicates with the server with packets through a network (i.e. using the network connector), and communicates with the client components with events through several event buses. Additionally, having to specify the direction of the connections—whether in, out, or both—further made us reflect on the flow of the events in the system. For example, many events are received from the server, but the only component in the client that generates events is the current player's lunar lander. This makes sense since the users can only control their own lander.

Overall we had a positive experience with xADL. The notation is very flexible with how much detail the model has. Since the UML model captured the details of the server, for the xADL model we chose to detail the architecture of the client and leave the server as a single component. However there is nothing stopping us from expanding its level of detail if necessary. On the other hand, xADL was also very simple. For all of its expressive power and ability to define new components, the graphical view boils down to components, connectors, and links. This meant that we were free to focus on the overall structure of the architecture without making too many implementation decisions. Of course the downside of staying at a high-level view meant that we could not perform very much analysis on the model. The model was still very valuable in organizing our thoughts on how events will flow through the system once it is implemented.

ArchStudio 5 was relatively pleasant to use once we got the hang of it, but it had many obstacles. The first one was the lack of documentation or tutorials. There was plenty of information about installing and setting it up, but hardly any resources on how to actually use it to build models. As a result, a lot of the work was trial and error and exploring around to understand the types of things ArchStudio could do. Archipelago worked well, but lacked some user-friendly features such as keyboard commands or predictable mouse gestures (e.g. selecting multiple elements by dragging a box around them, moving the scrollbars instead of zooming when using the mouse scroll wheel, etc.). Despite the shortcomings, ArchStudio was useful and allowed us to explore the design space of our application.

Lunar Lander Base Race Darwin Model



Using the Darwin modeling notation turned out to be a very pleasant and intuitive experience. The structural and easy-to-understand nature of modeling with Darwin were the reasons why we chose it as one of our models. We were unable to find a sufficient tool to model Darwin via the textual syntax; therefore we chose to proceed with the graphical visualization of Darwin. Our Darwin model is focused around the information shared by the classes and objects of Lunar Lander Base Race game client. By using the implementation of GameNet (as described in the UML model) we are able to have the client only with a lightweight GameNet connector which is shown in the Darwin model. The rest of our model shows how information is shared between objects—by either being provided or required—in the game by everything from the menu game setup to the in-game components.

Darwin was an enjoyable notation to work with and we felt it nurtured the development thought process. The concept of components in the game was where we began. After laying out most of the components we were able to smoothly transition into thinking about what information would be required by each component and who provide that information. The Darwin notation was very flexible in easily adding additional nodes to connect the components. We found it was also very easy to develop the model as a team because there was no doubt to the other developers on the team what each labeled node was as they were added, which was very intuitive to follow. Overall Darwin proved to be instrumental in gathering and mapping our thoughts and in the end we have a solid foundation in which to begin structuring our code.

As mentioned previously, we were unable to find any tools that catered well to Darwin so we constructed the graphical model using Microsoft Visio. Using this tool to construct a Darwin model was surprising a good fit. With a few simple shapes and reference links we were able to create a clean Darwin model. Additionally Visio made it easy to adjust moving components around as the flow and structure developed, all while without undoing the individual components setups. Due to its proven ease and easy readability we all look forward to bringing Darwin into future development for structure planning.

Consistency

UML Activity Diagram and xADL

The UML and xADL architectural models are complementary since they focus on different areas of the system. The architectural style of our system is an event-based architecture. The xADL model shows the overall structure but focuses on the client/publisher side of the architecture. The server is shown in xADL only as the “GameNet Server” component that sends and receives events. On the other hand, the UML model shows in great detail how the packets used to communicate events are received, interpreted, and sent from the server. In the client part of the system, components communicate with the server via different types of events. The table below shows the correspondence of event types in xADL (represented by connectors) with the decoding paths in the UML.

xADL	**UML**
Could not connect	Purpose 0 (Handshake)
Alias in Use	Purpose 0 (Handshake)
Game Creation	Purpose 2 (Create Game)
Game Joining	Purpose 3 (Join Game)
Game Management	Purpose 1 (Get Created Games), Purpose 5 (Set Game Joinable)
Connection Lost	N/A (By definition server would not know about this event)
Game Update	Purpose 7 (Game Update)
Player Join or Left	Purpose 4 (Left Game), Purpose 6 (Disconnect)

UML Activity Diagram and Darwin

There is very little overlap between the UML Activity Diagram and the Darwin model. This was done intentionally as we wanted each of the models to highlight different aspects of our system. The UML diagram focuses on the server's event bus and how it pertains to our system while the Darwin model focuses on the game implementation on the client. As shown earlier, the xADL model describes the overall system layout and includes the communication between the game client software and the server event bus.

xADL and Darwin

The xADL and Darwin models have some overlap as the xADL focuses on the whole system with an emphasis on the client side and Darwin is all client side. Both models depict the GameNet connector as the component that handles communication with the GameNet server. Additionally both models include the core game components, depicted on the table below.

xADL	**Darwin**
Main Menu	Create Game
Games Lobby	Game.Net Lobby
Game	Game
Created Game Lobby	Game Lobby
Environment	Environment
Player 1 Lander	Lunar Lander
Player 1 Orbiter	Orbiter
Player 1 Moon Base	Lunar Base
Player 2 Lander	Lunar Lander
Player 2 Orbiter	Orbiter
Player 2 Moon Base	Lunar Base

The main differences come from the different purposes of the models. In the xADL model each of the two players is represented separately to show the differences in their event flow directions. The Darwin model on the other hand focuses on the information both players

require and provide and thus shows more clearly how the components related to the player share that information.