GAME ARCHITECTURE

GAME CONCEPT

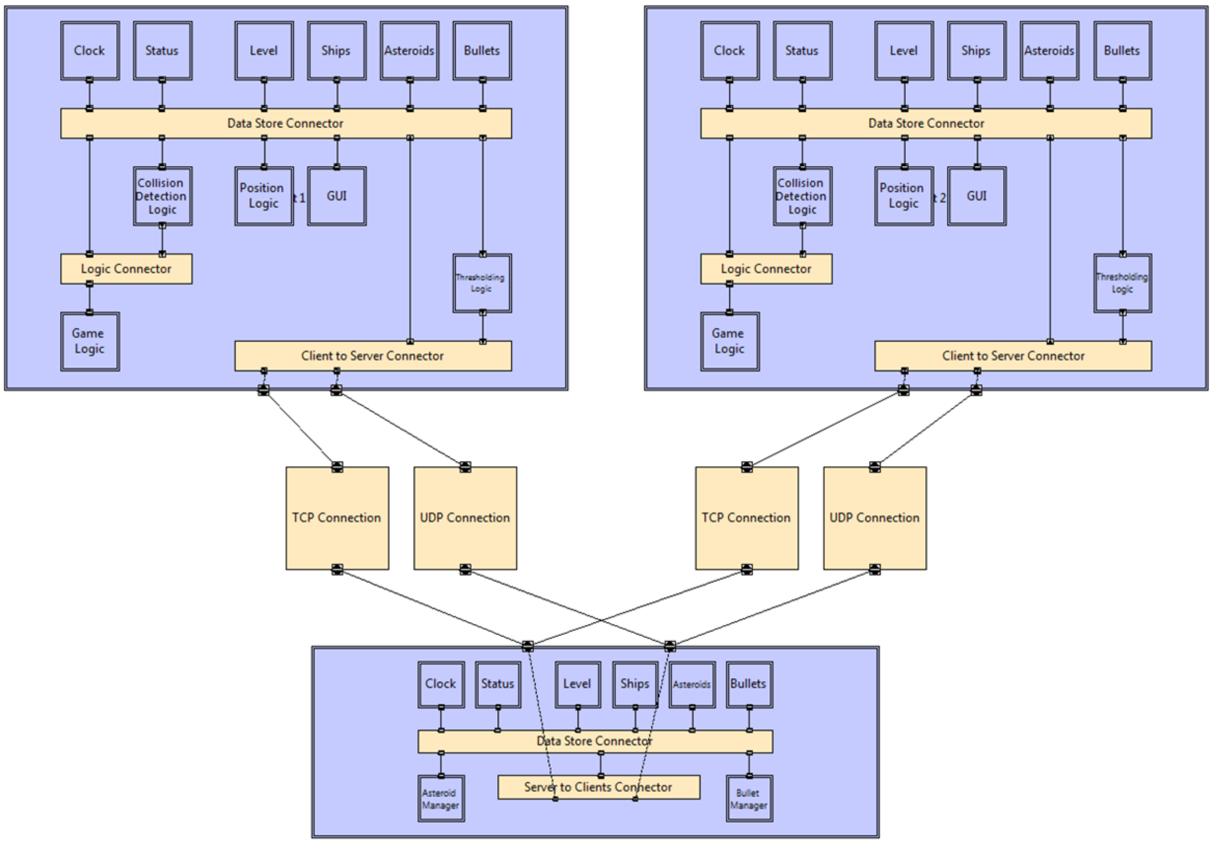
The presented game is a simple multi-player variant of the 2D game Lunar Lander[1]. Each player controls a separate ship. As additional obstacle, the players have to escape asteroids that fly over the moon surface. Their ships are also equipped with firearms, which can be used to destroy asteroids. Instead of fighting against each other, the players cooperate towards the goal of one of them safely landing on the moon surface. This could e.g. be accomplished by other players watching out for asteroids and clearing the way while one player tries to land. Players can shoot each other, but if one player's ship is destroyed, the remaining ones continue the game without that one, either until one ship successfully landed, or until all ships were destroyed.

MODELING LANGUAGES AND TOOLS

For the architecture description of the Lunar Lander game I chose the following three modeling languages / tools:

- xADL – to model the static game structure in a coarse-grained fashion, using Archstudio 4[2].
- ACME – for a finer-grained model of the networking connector structures, using ACMEStudio[3].
- UML – for behavioral models of the game initialization and GUI functionality, using Visual Paradigm[4].

COARSE-GRAINED STRUCTURAL GAME MODEL: XADL



The game is divided into a client and a server application. The above diagram shows two of clients connected to a server, all of them with their internal component structure. The game architecture does not pose any explicit limits on the number of clients simultaneously connected to the game server.

The connection between a client and a server is maintained over two channels / connections: a TCP connection and a UDP connection. While the UDP channel is used on the one hand to announce server availability (via broadcasts by the server) and on the other hand to exchange less important information (for example, ship movement updates which do not require a retransmission in the event of packet loss, because they will become obsolete with the next update), the TCP connection is established to monitor the client-server connection status and exchange important information (for example, ship or asteroid destructions and status changes).

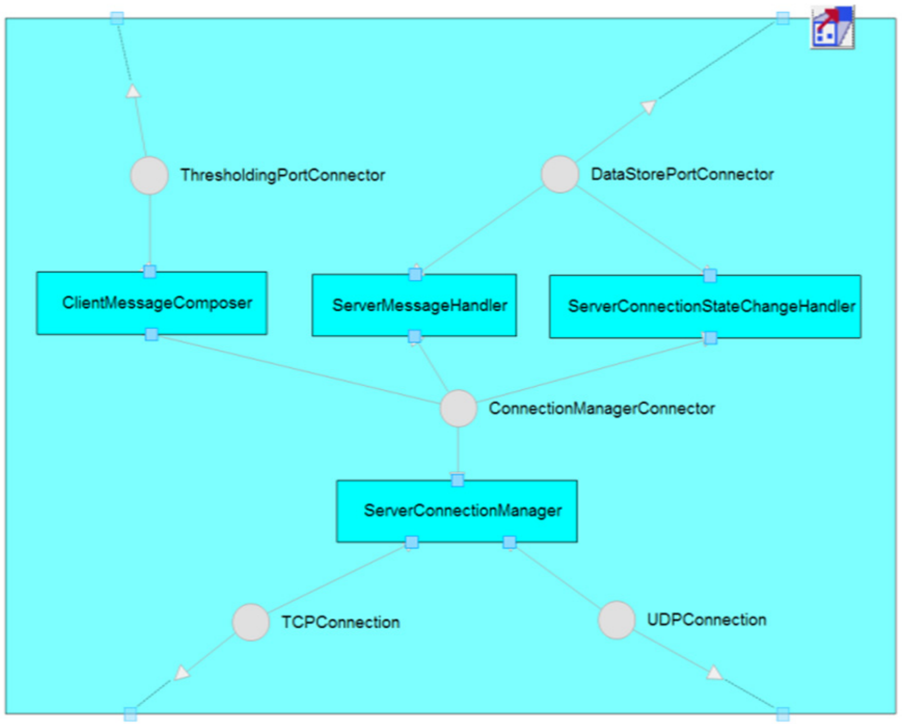
Both the client and the server implementation are based on the C2 architectural style[5] on the displayed granularity level, although the implementation of the single subcomponents (for example, GUI or Game Logic) does not necessarily conform to this style as well.

The uppermost level in both the client and server architecture consists of data store components (Status, Level, Ships, Asteroids, Bullets) and the game clock component. All of which are connected to the same connector (Data Store Connector). The data store components can process data store update requests from the components below them in the architecture, and will send data store update notifications downwards once they performed an update. The movement and position of moveable items of the game (ships, asteroids and bullets) is stored by a base position and speed vector for a certain time and an acceleration vector, to decrease the necessary network traffic for position / movement changes.

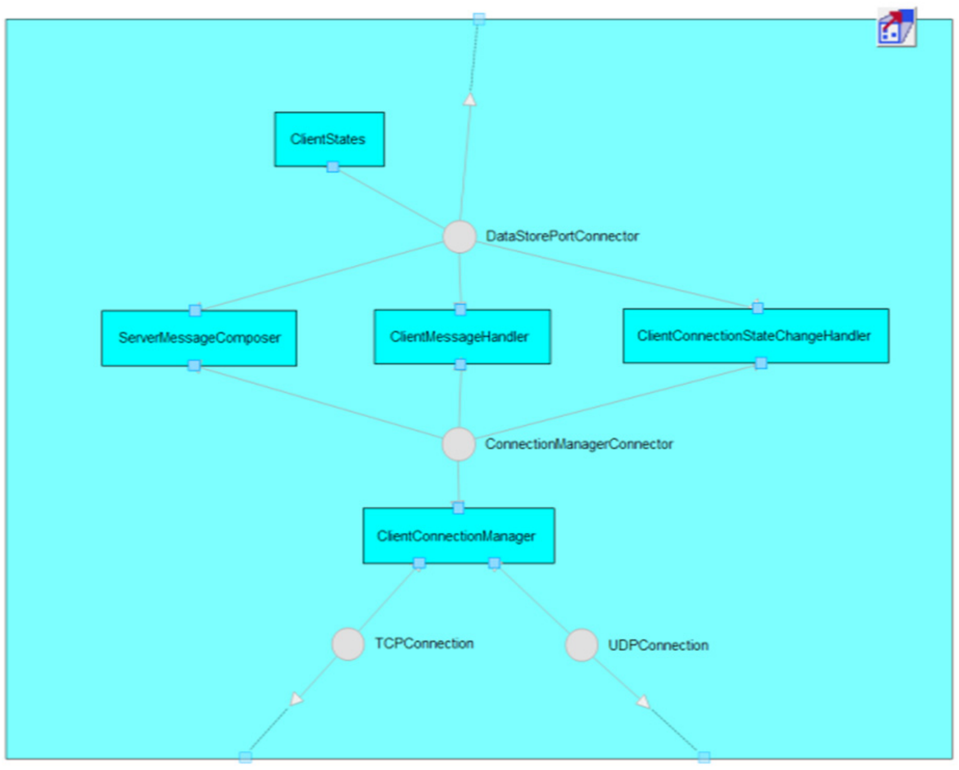
In the server architecture, the Data Store Connector connects the data store directly to the connector that handles the client-server communication (Server to Clients Connector), as well as to the components that manage Asteroids (creation and destruction on out-of-world movement) and Bullets (destruction). Other than that, the server mainly functions as a data store only, forwarding requests and notifications between the clients and managing the client connections and their data consistency. (For some more details see the game initialization model further below).

In the client architecture, the Data Store Connector connects to the Collision Logic, Position Logic and GUI components to the bottom, as well as to the Logic Connector and the Client to Server Connector, whereby update notifications to the Client to Server connector are first processed by a Thresholding component before they are passed on to the connector. The Position Logic calculates the current position and speed of ships, asteroids and bullets, by extrapolating from the last position of the items and their last speed and current acceleration vectors and writes the results back into the data store (by sending the respective update requests). The GUI component paints the user interface based on the data store state and sends update requests to it based on the user input (for example, changes in the ship acceleration or new bullets). The Thresholding component is responsible for only passing on data store changes to the Client to Server Connector when the change is bigger than a given threshold for the specific type of the change (for example, only pass on ship movement changes if the movement deviates significantly from the predicted path from the last sent position, speed and acceleration vectors). Collisions between Bullets, Asteroids or Ships will be detected by the Collision Detection component and passed on to the Logic Connector, which in turn passes on notifications form the data store and Collision Detection to the Game Logic component. This component will react upon possible collisions or state changes and evaluate their effect, issuing possible data store update requests.

STRUCTURAL MODELS OF THE NETWORK CONNECTORS: ACME

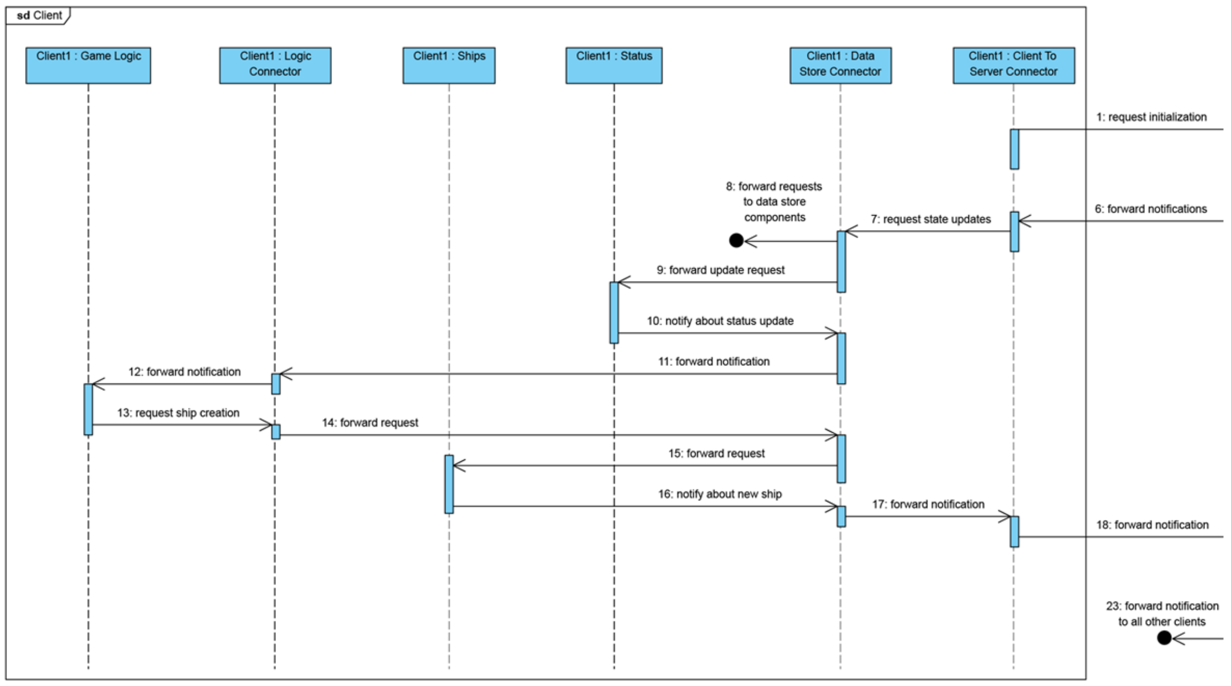


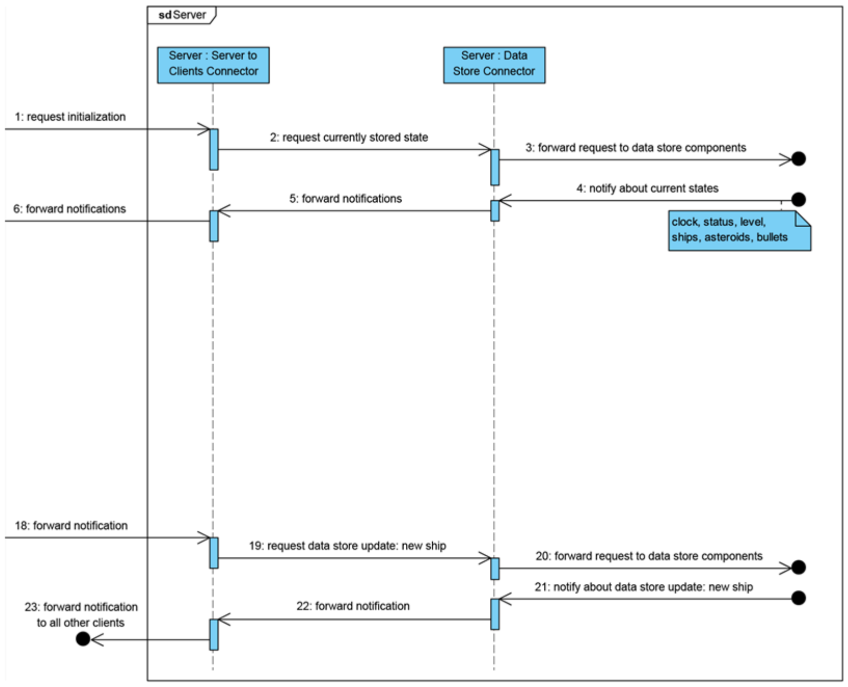
The above diagram shows the internal structure of the Client to Server Connector present in the client architecture as shown in the xADL model. Its architecture also follows the C2 style. The Server Connection Manager is responsible for serializing and deserializing messages, and transmitting / receiving them over the TCP and UDP connections to / from the server. In the case of arriving messages, it sends a handle message request to the Server Connection Manager Connector, which will forward it to the other three components. The Server Message Handler will then analyze the message and issue data store update requests to the Data Store Port Connector, which forwards them to the Data Store Connector. If the connection status changes, the Server Connection Manager requests the handling of the state change, which will be served by the Server Connection State Change Handler. For example, if the Server Connection Manager signals a connection loss, the state change handler will request a game status update respectively. Contrary, if any data store update notifications arrive at the ThresholdingPortConnector, then those will be handled by the Client Message Composer and the Server Connection Manager Connector will be notified of a new message ready for transmission (which the Server Connection Manager then sends).



The above diagram shows the detailed architecture of the Server to Clients Connector present in the Server architecture in the xADL model. It is analog to the Client to Server Connector architecture depicted above, but includes an additional Client States component, which stores the current connection states to the clients. These are updated by the Client Connection State Change Handler and the Server Message Composer can use the notifications about their change (sent by the Client States component) to broadcast the state change to all other clients. The Client Connection Handler has similar responsibilities as the Server Connection Handler in the Client to Server Connector architecture, but handles more than one connection simultaneously and provides a TCP connection / socket that the clients can connect to. The Client Message Handler evaluates the incoming messages and requests data store updates. The update notifications will be handled by the Server Message Composer and broadcasted to all other clients.

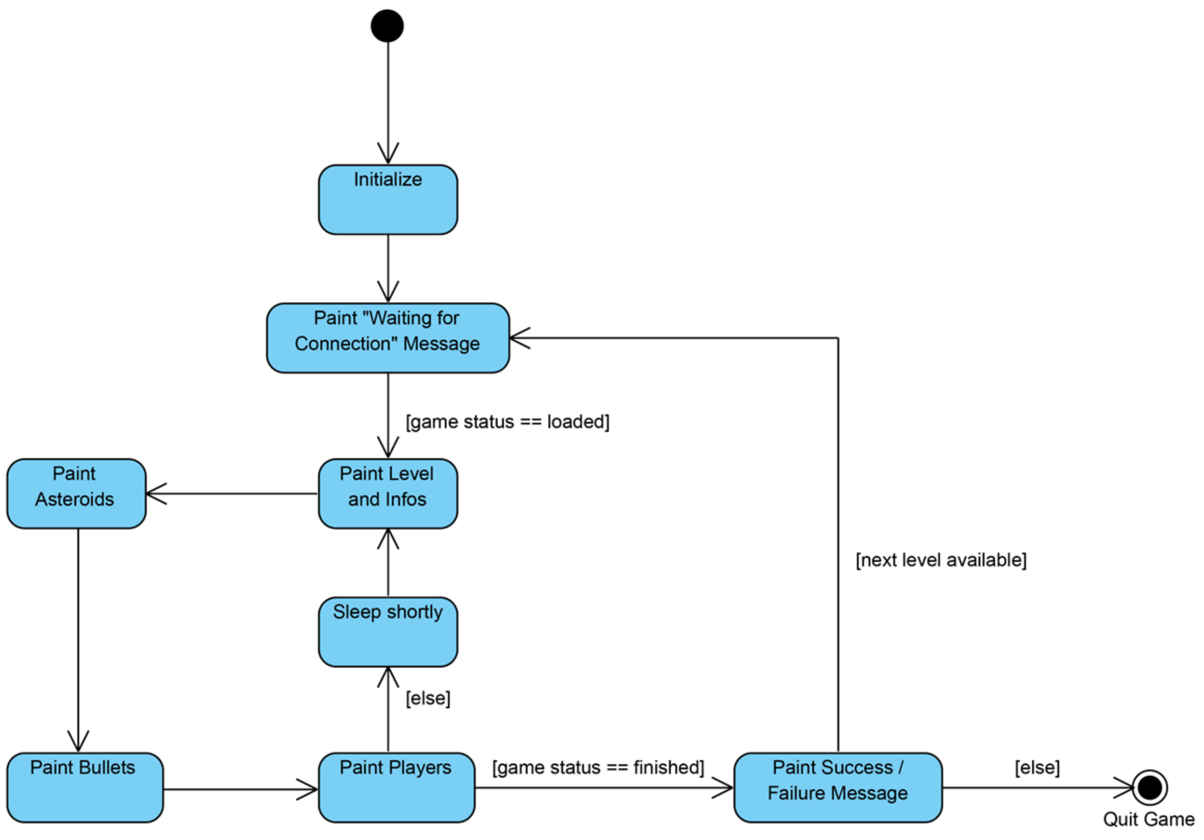
BEHAVIORAL MODEL OF THE GAME INITIALIZATION: UML


sd Client
Client1 : Game Logic
Client1 : Logic Connector
Client1 : Ships
Client1 : Status
Client1 : Data Store Connector
Client1 : Client To Server Connector
1: request initialization
6: forward notifications
7: request state updates
8: forward requests to data store components
9: forward update request
10: notify about status update
11: forward notification
12: forward notification
13: request ship creation
14: forward request
15: forward request
16: notify about new ship
17: forward notification
18: forward notification
23: forward notification to all other clients


sd Server
Server : Server to Clients Connector
Server : Data Store Connector
1: request initialization
2: request currently stored state
3: forward request to data store components
4: notify about current states
clock, status, level, ships, asteroids, bullets
5: forward notifications
6: forward notifications
18: forward notification
19: request data store update: new ship
20: forward request to data store components
21: notify about data store update: new ship
22: forward notification
23: forward notification to all other clients

The above sequence diagram (splitted for better display) shows the client initialization process. After a connection to a server has been established, the client requests the initialization data, which the Server to Clients Connector obtains from the data store and sends back to the client. The Client to Server Connector then requests data store status updates locally, which in turn trigger the creation of a ship for the player (initiated by the Game Logic). The notification about the new ship will be serialized and transferred to the server by the Client to Server Connector. This process has been modeled exemplarily for other kinds of state exchange between the clients and the server.

BEHAVIORAL MODEL OF THE GUI FUNCTIONALITY: UML



The above state machine diagram illustrates the GUI rendering process. After initialization of the component, a message is shown to notify the user that the game is waiting for the connection to the server. Once the game is loaded, the GUI rendering process enters its main loop (paint level and information display, asteroids, bullets and players). After each cycle through the loop, the game status is checked and if the game has finished, the GUI paints a message notifying the user about the game outcome. When a next level is available, the GUI reenters the loop through the waiting-for-connection message state. Otherwise, the game is complete and will exit.

This diagram does not show any of the input processing that the GUI component has to implement as well. This is because the input processing can be implemented completely separately from the here depicted rendering process (in another execution thread).

CONSITENCY BETWEEN MODELS

All models are based on the concept of components and connectors. The xADL model functions as a reference model for all other models, as it describes the overall coarse-grained application architecture and all other models refine certain parts of this architecture. We will use it therefore to show the other models' consistency with this model.

In ACME, components and connectors can be modeled explicitly, similarly to xADL. They communicate through ports (components) and roles (connectors), which is little different to xADL's interfaces. The ACME models above define refinements of the client/server connectors in the xADL model. To allow modeling the substructure of the client/server connectors, they each had to be modeled as a component in ACME. Nevertheless, they provide and require the same interfaces / ports (Client to Server Connector: to TCP, UDP, Data Store, Thresholding; Server to Clients Connector: to TCP, UDP, Data Store). Thus, the refinement these models describe fits well into the xADL model.

The UML sequence diagram does not feature an explicit modeling of components and connectors as language-level types. Thus, they have been modeled as common actors in the sequence diagram and named respectively. The sequence diagram only shows message / event passing in the C2 request/notification style, and only between components and connectors that have connected interfaces. The granularity of the interactions / component boundaries corresponds to the granularity in the xADL model and abstracts from the refinements in the ACME model (which does not pose any inconsistency, as the sequence diagram shows another level of abstraction and could be refined to provide this information as well). Thus, the models are consistent.

The UML state diagram shows the refinement of the operation of a single component in the xADL model as state transitions of the component (behavioral, not structural) and thus does not need to provide support for component or connector notations. It conforms to the xADL structure in that it only supposes knowledge about information that the GUI component could acquire through notifications from the data store.

MODELING ASSESSMENT

GENERAL ASSESSMENT OF SOFTWARE ARCHITECTURE MODELING

As main take-away from this exercise, I consider the insight that modeling architecture can be long, painful process, but also a fruitful process, even for a game of this small scope. The explicit modeling makes you think carefully about the features, necessary component boundaries and relationships, and thus can reveal thought problems that otherwise would have only surfaced someway into the implementation. In this case, for example, the modeling forced me to think of a way to model the movement of items in the game in a way that was easily transferrable over the network without causing too much traffic, thus preventing a possible less thoughtful implementation. It was definitely a worthwhile experience.

Nevertheless, I am unsure whether architecture (at least in the requested formal level and detail) is a good investment for a project like this (in "real" life), as for a project of this scope, apart from the learning effect, the only take-away from the architecture is the possible speedup in the implementation gained through a well-defined architecture. Other benefits of software architecture will not be used (for example, documentation for later project development/maintenance, component reuse, analysis or similar). Thus, the cost of going the wrong way in the implementation (for example, as usually pursued in extreme programming techniques), might not be outweighed by the speedup gained through the use of architecture models (given the big amount of time it takes to create the architecture). I stay skeptical, and look forward to the insights gained through the implementation of the game based on the architecture models.

LESSONS LEARNED DURING THE MODELING PROCESS

Looking back on the obstacles I had to deal with and insights I gained during the modeling process, the first thing to note is probably that this assignment had a very slow and difficult start. Especially if you aren't very familiar with real time networked game development on the one hand and the architecture tools on the other hand, being productive from the start was very hard. Without a specific idea about the implementation architecture, you cannot really start modeling. So apart from learning about the tools, you also need to start by researching common problems regarding design decisions with networked games. Thus, it was a very time-consuming task.

Apart from this, another obstacle was finding ways to make the different modeling notations compatible (i.e. enforcing the same component and connector boundaries in all languages). For example, the hierarchical composition of components in ACMEStudio did not easily allow connecting connector roles to ports of outer components (see tools section). Also, maintaining the consistency between the models in the different tools was difficult, as this was only possible by manual inspection and change. This definitely created desire for the idea of modeling all different aspects and diagram types in a single tool that can make syntactic and semantic crosschecks between them (e.g. references to same entities in all models) and possible even allow changes in one model to propagate to other models.

MODELING LANGUAGES

xADL 2

Comparing xADL with the other languages, the indifference between interaction of connectors and components in xADL (both have a set of interfaces and can be connected to either type) made a positive impression. xADL provided a direct notation of components and connectors, thus making it easy to put the architectural thoughts (which, in my case, conformed to hierarchical components and connectors) into a formal model. On another note, I found the types / structures concept of xADL 2 very intuitive for creating substructures, even though it was removed in xADL 3. Furthermore, one can safely say, that the actual XML representation of the model cannot be edited without tool support like Archstudio (the final document was about 4000 lines long). The extensibility features of xADL were not used for this project, they simply were not necessary for its scope.

ACME

ACME provides an easy and readable textual syntax, and models can be easily created without tool support. Nevertheless, there is a graphical editor for ACME included in ACMEStudio, which makes model creation even more comfortable. The language provides extensibility features, including adding properties to component / connector types and style families to create new types of model entities. The editor also provides the necessary configuration points for their representation (see next section).

For this project, the entity set did not need to be extended, but one feature that was missing in the default language entities was the direction of the ports (in / out / inout) and had to be modeled as property. Those sadly cannot easily be displayed in the visual representation.

UML

The UML sequence diagram does not notationally support the architecture concept of components and connectors, which immediately shows the fact that this type of diagram was not specifically designed for architecture modeling in the granularity I used it for. Nevertheless, the loosely defined syntax and semantics of UML sequence diagrams allowed me to find a way to model everything I expected. Components and connectors were both modeled in the same way, the message passing semantics provided enough room to support the coarse-grained level I used, and even modeling messages to several components at once could be modeled with lost and found messages. Nevertheless, it would have been nice to have syntactical support for all of the above, so as to reduce possible misinterpretation of the diagram representation.

The UML state diagram could be easily used to refine a single components behavior, as no interference with other components is explicitly modeled in it. While this may be an advantage on the one side, on the other it also is a disadvantage because inconsistencies between the models (for example, the use of data in the state diagram transitions that cannot be obtained through the static/structural architecture) cannot easily be checked for and even manual inspection requires effort.

Concluding, UML allowed a very fast modeling because of few syntactical constraints, and the wide usage of UML in the industry comes with very good tool support. Its usage can serve the purpose of building a consensus of the understanding of the architecture, but it requires explanation and discussion of the models to obtain it, given the ambiguity of its notations regarding architectural models.

MODELING TOOLS

Archstudio

I decided to use Archstudio 4 for the modeling in xADL. There were several factors that contributed to this decision. In the lecture, this version was promoted as more “stable”, and supporting the 1.x mapper tools (which I would like to try out). Also, I gave Archstudio 5 a fast shot, and couldn’t figure out how to do the hierarchical compositions without component types at once (I didn’t realize the drag and drop features and could not find any context menu entries for this in Archipelago neither any how-to that would explain this. Personally, I found working with the types concept very intuitive in Archipelago in Archstudio 4, although mapping signatures and interfaces was a little cumbersome because it felt like defining them twice). Thus I went with the older version, especially as it seemed to still be well maintained and working on top of Eclipse Juno. All experience described below refers to Archstudio 4.

The context-menu based interaction in Archipelago was easy to understand, although a full tutorial on hidden features would have been nice. I would have loved some keyboard shortcuts (for example, F2 for renaming entities, or some key combination for creating new interfaces on components) and more mouse actions (for example, ctrl-click-and-drag or something similar for creating connections between interfaces). I experienced some issues with drawing exactly vertical/horizontal links, which were promptly fixed on request. Other than this, I would like to propose an improvement for changing interface directions: If two interfaces are connected, changing the direction of one interface could possibly also change the direction of the other interface, as the connection would not make much sense otherwise. For example, if one interface is changed to “in”, the other end of the connection should be an “out” interface. Of course, if more than one connection to the interface exists, other possibilities might have to be considered. Another improvement, which is probably only relevant for the typing concept of xADL 2, is to add options for creating interfaces from the signatures of the component types or vice-versa. Something else I really missed was undo/redo functionality in Archipelago.

Despite all these possible improvements, Archstudio and Archipelago were quite easy to get comfortable with and very stable. My overall impression was a positive one.

ACMEStudio

As mentioned before, the editor in ACMEStudio provides a very good configurability of the representation of entities and entity types. For example, it was easy to add a label to the connector representation. Especially given the extensible nature of ACME, this seems like a nice feature. Another positive impression was the easy possibility to switch between source code and the graphical editor through editor tabs, although the source code representation inside the editor was sadly very buggy as soon as component hierarchies were introduced. Which introduces the first and foremost issue with ACMEStudio: It is buggy and has a very low responsiveness (it is slow!), especially if the model gets a little larger. Also, the graphical editor did not allow binding interface roles to outer component ports in a hierarchy (as described above), although the language supports this.

Concluding, I would like to say that even though the website for ACMEStudio seemed only half-way filled, there were nice PDF tutorials available which provided a good introduction and made using the editor feel relatively intuitive – I did not have to search for long for features. I would not use the editor for any productive projects, but it showcased a few nice ideas worth exploring.

Visual Paradigm

I should say that I had experience in working with Visual Paradigm before, and thus my impression might look better than it once was. I can remember that I had a very fast start with this tool when I first used it. The controls are very intuitive and fast. Most actions can be accessed via buttons that show up on mouse-over / selection of entities in the diagram directly next to the entity. Making connections between entities by drag-and-drop of those buttons is very simple. Some features are a little hidden in property dialogs; for example, how to change the way that message sends are numbered in a sequence diagram was not easy to find out. Also, the tool is not bug-free either – for example, activity markers on a sequence diagram lifeline happen to change length sporadically without apparent reason. Nevertheless, one can easily recognize that Visual Paradigm is a commercially deployed product, and has matured well.

GAME / SYSTEM PROPERTIES

As already mentioned before, one big thing I learned about the properties of the system was how to model the state of the moving items in the game in a way that was easily transferrable over the network. Apart from this insight, modeling the client-server connector's inner structure showed that both a UDP and a TCP connection between each client and the server would be advisable. Through the model of the game initialization phase, I could clarify the relationships between the individual components and the way information would get from the clients to the server and vice-versa.

REFERENCES

[1] Atari “Lunar Lander” online arcarde game. http://atari.com/arcade#!/arcade/lunarlander/
[2] Archstudio 4. http://www.isr.uci.edu/projects/archstudio-4/www/archstudio/
[3] ACMEStudio. http://www.cs.cmu.edu/~acme/AcmeStudio/index.html
[4] Visual Paradigm for UML. http://www.visual-paradigm.com/product/vpuml/
[5] R. N. Taylor, N. Medvidovic, and E. M. Dashofy. 2009. Software Architecture: Foundations, Theory, and Practice. Wiley Publishing.