NTNU

Grumpy Old Git Club
Team Number: 16

TDT4240 Software Architecture

Castle Crush
ATAM evaluation
`define(COTS, ["Android", "Google Play Services"])`

Cabral Cruz, Samuel (496704)
Claessens, Bart (486346)
Ihlen, Erling Hærnes (765137)
Trollebø, Jarle (766901)

Primary Quality Attribute:
Modifiability
Secondary Quality Attribute(s):
Usability and Availability

Presented to
Alf Inge Wang

Department of Computer and Information Science
Norwegian University of Science and Technology
(NTNU), Trondheim
Spring 2018

Contents

List of Tables

Introduction

In the context of the development of a multiplayer game for the course TDT4240 - Software Architecture, we have been asked to proceed to the Architecture Tradeoff Analysis Method (ATAM) evaluation of another team's architecture. We have been assigned to the *Group 15* and the name of their game is *Castle Crush*. Their team is composed of the following members:

- Elgaaen, Truls
- Johansson, Ludvig Lilleby
- Kjernlie, Erik
- Mortensen, Jørgen Bergum
- Wahl, Nina Marie

For their game, the development team decided to focus on the **Modifiability** as their primary quality attribute. The team also considered the **Usability** and **Availability** quality attributes during the *Requirements* and *Architecture Design* phases of the Software Development Life Cycle (SDLC).

Game Concept

Briefly speaking, the concept of the game *Castle Crush* can be described as a turn-based game into which the players have to destroy the castle of their opponent using projectiles fired using a cannon placed next to their castle. The game is supposed to allow for different game mode but the main focus will be put on the multiplayer mode. This game is planned to be ported on Android Platforms and their application will relate on Google Play Game Services (GPGS).

To reach their goals, the team agreed on the selection of the following tactics and patterns:

Tactics

- Modifiability
  - Module Splitting
  - Increase Semantic Cohesion
  - Refactor
  - Encapsulate
  - Restrict Dependencies
  - Use an Intermediary
- Usability
  - Support User Initiative
  - Support System Initiative
- Availability
  - Ping/Echo
  - State resynchronisation, Passive Redundancy

Patterns

- Model-View-Controller (MVC)
- Singleton
- Template Method
- Observer
- Peer-to-Peer (P2P)

ATAM Description

The ATAM evaluation method has been developed around the concept of prioritized quality attributes. As one of the only input needed by the ATAM process, the quality attribute scenarios will be used to create a list of issues and concerns about the correctness of the architecture evaluated. One of the main advantages of the ATAM method is its flexibility. This flexibility can be seen across different aspects. Among them, we can find that the evaluation team does not need to be familiar with the architecture or the business goals and that the considered system does not have to be implemented. [Bass, 2013, Chapter 21, Section 2]

Document Structure

In this document, the details of the ATAM process for the game *Castle Crush* will be presented. By following the different steps of the process, we will start by introducing the Attribute Utility Tree. Then, an Analysis of Architectural Approaches will be provided. From this analysis, a set of **??**, **??**, **??** will be extracted. Finally, we will end this document with a short discussion on Own Experiences from Using ATAM. As usual, a list of Issues and a log of the Changes will be available at the end of the document.

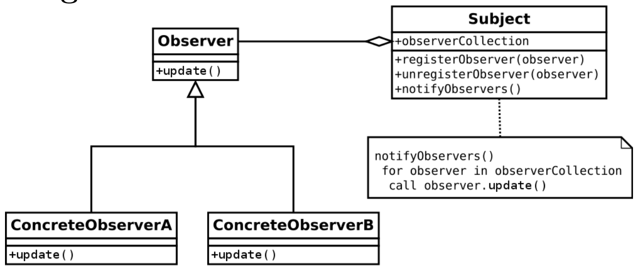
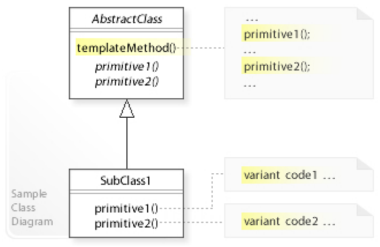
Attribute Utility Tree

Attribute	Scenario	Priority	Details
Modifiability	Change graphics	M - L	Within 1 hour
Modifiability	Add in-game objects	H - L	Within 3 hours
Modifiability	Modify in-game objects	M - M	Within 30 minutes
Modifiability	Add new level designs	H - L	Within 3 hours
Modifiability	Modify existing level design	M - M	Within 1.5 hours
Usability	Start new game	H - L	Less than 3 clicks
Usability	Complete tutorial	H - L	Within 2 minutes
Usability	Change settings	H - L	Within 20 seconds
Performance	Response time	M - M	Within 50 ms
Availability	Unplanned downtime	H - H	Notified within 1 minute
Availability	Gaps internet connection	H - H	Notified within 10 seconds

Table 1: Attribute Utility Tree for the game *Castle Crush*

Analysis of Architectural Approaches

Modifiability

ID: M1	**Scenario:** Add different game levels/designs			
Attribute(s):	Modifiability			
Stimulus:	Wish to add different level designs			
Response:	Make changes, unit testing, possibly user testing. In 3 hours, levels will be integrated in the system without adapting other modules			
Architectural Decisions	**Sensitivity**	**Tradeoff**	**Risk**	**Non-risk**
Observer	S4		R3, R4	
Template Method		T2	R5	
Reasoning: ▸ Observer pattern will be used to notify the other bricks constituting the castle after the destruction of a brick for the repercussion on the remaining structure. ▸ Template Method pattern could be used to define the generic initialisation of the world by varying the procedures to be used to incorporate new game elements or induce more randomness per example.				
Architectural Diagram:  				

```
notifyObservers()
 for observer in observerCollection
  call observer.update()
```

Table 2: Add different game levels/designs

Availability

Availability will be about detecting and preventing faults, as well as be able to recover from them.

ID: A1	**Scenario:** Detecting and recovering from player losing connection			
Attribute(s):	Availability			
Stimulus:	Normal operations			
Response:	One player looses connection. Return to main menu with an error message that the opponent is disconnected.			
Architectural Decisions	**Sensitivity**	**Tradeoff**	**Risk**	**Non-risk**
Ping/Echo	S1		R1	
P2P Connection	S2, S3			N1

Reasoning:

- Ping/Echo ensure that the game participants will know when a user is no longer connected to the game.
- P2P will ensure that the only connected players can do the ping directly at participants, reducing the need for a server to notify it for you.

Architectural Diagram:

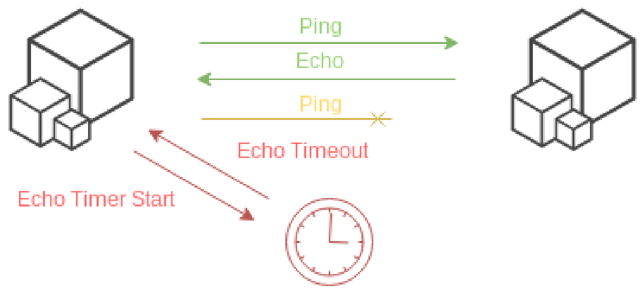


Table 3: Detecting and recovering from player losing connection

ID: A2	**Scenario:** Detecting players' game state out of sync and recovery			
Attribute(s):	Availability			
Stimulus:	Normal operations			
Response:	One player's game state update is not notified to other participants. Use game state that was last accepted by both devices.			
Architectural Decisions	**Sensitivity**	**Tradeoff**	**Risk**	**Non-risk**
P2P Connection	S2, S3			N1
State Resynchronisation		T1	R2	

Reasoning:

- P2P connection will create a channel for where the clients can share their latest accepted game state
- State resynchronisation will assure the clients that the state they are using, is the latest agreed game state, so that nobody will play on a different game state than their opponent

Architectural Diagram:

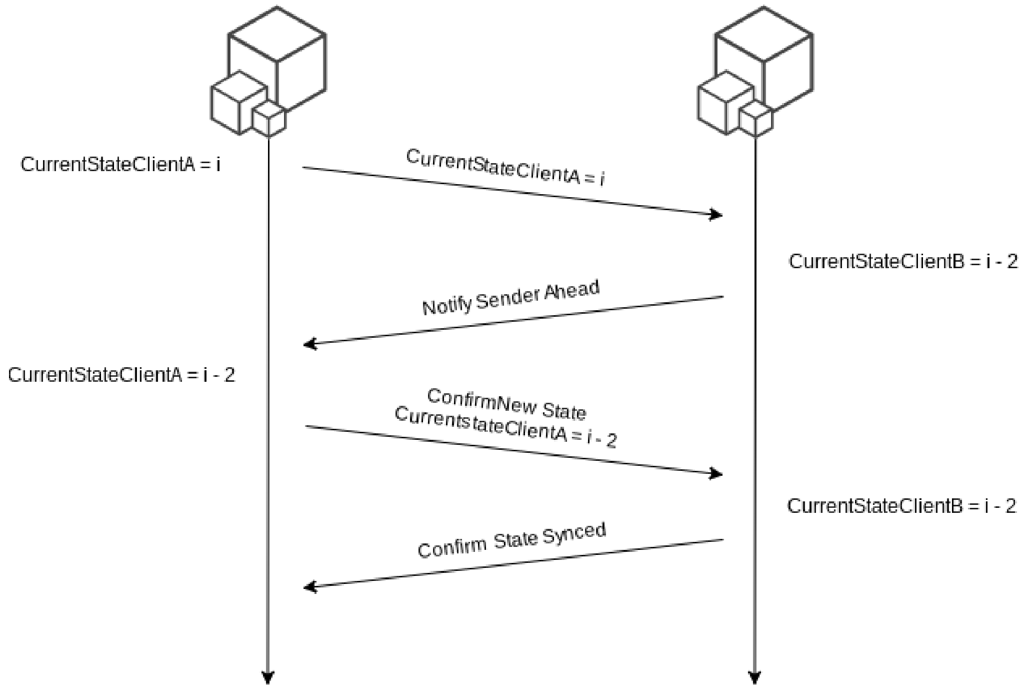

CurrentStateClientA = i
CurrentStateClientA = i
CurrentStateClientB = i - 2
Notify Sender Ahead
CurrentStateClientA = i - 2
ConfirmNew State
CurrentstateClientA = i - 2
CurrentStateClientB = i - 2
Confirm State Synced

Table 4: Detecting players' game state out of sync and recovery

Discussion

Sensitivity Points

S1 Ping/Echo will have a positive effect on availability

Ping/Echo improve the availability by minimizing the down-time caused by any failure. The Ping/Echo will normally be at the head of the recovery protocol since this tactic cannot prevent a failure by itself.

S2 P2P connection will have a positive effect on availability

The availability is enhanced in a P2P connection because the devices concerned and having an interest in the connection will be their own provider of the processing power needed. In a traditional client-server connection, the burden would normally be completely delegated to the hosting servers which would normally necessitate a lot more processing power, a complex hardware architecture and a bigger overhead for the sharing of the ressource among the different requests received by the server.

S3 P2P connection will have a positive effect on performance

By using a P2P connection, the data package can be transfered directly from one player to the other without relying on the server has a communication intermediate. Hence, the number of data package transfered is roughly diminished by a factor 2.

S4 Observer pattern will have a positive effect on the modifiability

The use of the observer pattern will reduce the coupling between distant and unrelated classes and increase the semantic cohesion at the same time. One could argue that the observer pattern actually introduce more coupling, but this is done through the implementation of a `Listener` interface which we do not consider as a binding architectural decision. The observer pattern would definitely have a positive impact on the architecture by simplifying the communication flow between the different game objects.

Tradeoff Points

T1 State resynchronisation will have a negative effect on performance, but positive effect on availability

Having a state resynchronisation will be beneficial for the availability since it will provide a protocol to recover in case of fault if one of the players do not receive one or many messages and get out of sync. Recovering from this situation may however be processing expensive since a negotiation between all players must be performed in order to determine on which to agreed on and to perform the transfert of the necessary information to all the unsynchronized nodes.

T2 Template Method Pattern will have negative effect on modifiability, but positive effect on modifiability

As surprising as it can be, the evaluation team decided to classify the usage of the Template Method pattern as a tradeoff since we do not consider the this will help achieve the kind of modifiability required by the game dynamic.

The Template Method pattern is really useful when we want to make a basic algorithm

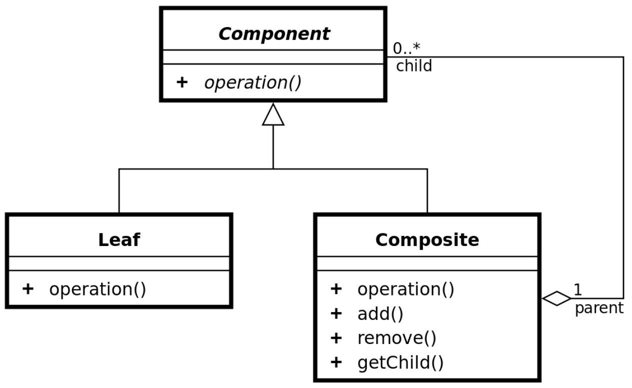
more flexible. This will usually be necessary when we want to create an interface having different options for transforming and/or interpreting the inputs. If we push it to the limit, we could simply describe the Template Method pattern as the application of polymorphism to procedural code generation. Wikipedia contributors [2018c]

However, the kind of modifiability that we are seeking for is much more related to

the object behavior itself more than the different subprocesses that we will have to be run to achieve a given result. Using the Template Method pattern may even introduce some undesirable constraints into the design.

A better pattern to be used considering the game concept would be the composite

pattern where a given interface has to be implemented by the whole class hierarchy which allows us to solve recursive problem such as the construction of a castle and other game elements. The main idea behind the composite pattern is that every object can become a combinaison of sub-object and every time a given method will be called, the main object will simply recursively call the same method over all the objects that he is composed of. Based on this idea, we could then define the castle as a composition of basic shape of blocks. Wikipedia contributors [2018a]



Risks and Non-Risks

R1 Ping/Echo can introduce unnecessary complexity

No real value will be added to the implementation of a Ping/Echo protocol because the connection between players will still be lost and the game will have to stop. The only advantage we could see here would be the redirection of the connected players to the main menu windows whereas the player having lost the connection would face a freezing window in the worst case. By looking at the GPGS real-time multiplayer documentation, we can see that an event notification service is already provided by the framework. The documentation also mentions that the resulting connection set will be a set of fully connected players and that if any connection is loss between one player and another, the connection set will result in the remaining fully connected players network. [Google Play Game Services, a, Connected Set]

Finally, an analysis and a lot of testing could be required to determine the value to which the timer must be set to actually consider a player has lost its connection. In fact, having a small buffer can sometimes be useful for temporary disconnection.

R2 State resynchronisation will be hard to implement

Considering the low amount of data to be exchanged the need for unreliable and risk for asynchronism is not worth. Due to the turn-based aspect of the game, the reliable messaging option offered by the real-time multiplayer gaming session would be a better alternative than trying to implement a state synchronisation procedure.

> With reliable messaging, data delivery, integrity, and ordering are guaranteed. [Google Play Game Services, a, Sending Game Data, Reliable Messaging]

Furthermore, more investigation could be done through the documentation of the turn-based multiplayer gaming session. However, at the read of the description of their definition of a pure turn based game, the evaluation team was unsure that this will fit all the needs of your game concept. Google Play Game Services [b]

R3 Observer pattern may lead to memory leaks if not correctly implemented

One of the downside of the observer pattern is that it can easily lead to memory leaks and especially when the publishers can be subscribers and vice-versa conjointly. This is why great care should be used when it will come the time to delete (destroy) blocks if you decide to use this pattern in the final design. Wikipedia contributors [2018b]

R4 Observer pattern may be hard to implement over the `LibGDX`'s physics engine

Observer pattern usually comes in pair with Entity-Component-System (ECS) pattern. So if one decide to use to create its own observer pattern for the development of a game, he is merely forced to implement the ECS pattern along because it is usually the role of the physics engine to notify the objects that has to be notified. On the other hand, an alternative would be to register to the actual physics engine collision publisher to induce the change desired in the gameplay, but again it would results in duplication of work and processing. The best solution would be to use `LibGDX.Box2DPlot` instances for the bricks of the castle and to rely on the implemented physics engine or to go ahead and implement your own ECS and the observer pattern at the same time.

R5 Template method may barely suffice to introduce enough class hierarchy flexibility

Considering the repetitiveness of the gameplay, it will be important for the architecture to be flexible enough to include as many different variations of the basic game as possible. As demonstrated here, template method may not be sufficient to achieve this requirement.

N1 Establishment of the P2P connection can be handled by the Google Play Games Services

Considering the small amount of time at our disposition for the development of the game, it is actually a plus-value to relate on such a generic, well-documented and standardized interface to establish the P2P connection between the players and the room creation.

Own Experiences from Using ATAM

In this chapter we will describe how it was to use ATAM to review and discuss the architect of the other group.

What We Uncovered

During this process, we were able to uncover the fact that the other group's choice of wanting to implement state resynchronisation, and addressed how this will be very complicated and possibly affect the performance of their system. In uncovering this, we were able to let them know how this is a risk, and gave them the opportunity to reconsider their approach to this issue in their architecture. By using ATAM for the other group, we were able to uncover issues and potential risks about their architecture before they even started the programming, resulting in possibly a lot of hours saved in frustration when something could have broken.

We also made some recommandation regarding their main quality attribute, modifiability. By comparing the game concept with the selection of deisgn patterns, we have been able to conclude that the Template Method pattern would not lead to the desired results and that the Composite pattern would be a better alternative.

Thoughts About ATAM

As discussed in the previous chapter, using ATAM we were able to let the other team know of the risks their architecture could contain in trying to achieve their attempts to support their quality attributes to meet the business goals. The benefit of using ATAM here was to discover issues early, which is one of the main points of this procedure.

Even though we were able to use the ATAM process to achieve this, we didn't feel like the process itself was that relevant for this. When working on a relatively small project like this, we didn't feel like we benefited that much from it. We believe using ATAM and benefit from it will be a lot more relevant when working on way larger projects, and have stakeholders that have technical competence in what field the software will be used participating in the process.

Even though we were able to notify the other group of risks using this process, we believe this sort of risk would've been uncovered anyway if they had looked further into the offers they can get from GPGS, as they already have solutions for syncing and detecting disconnected players.

We agree with the benefits of discovering flaws in the architect before starting the coding, as this will save a lot of time and frustration, but we believe if we were to develop a bigger system, and cooperate with other participants than just being on our own developing a game from scratch, we might have been able to benefit more from ATAM.

Issues

No real issue was faced during the redaction of this report or the exercise of the ATAM evaluation. We want to show our gratitude to the development team of the game *Castle Crush* for having provided all the necessary information to proceed to this analysis in-depth and we hope our investigations will help them to improve their architecture and to successfully implement their game.

Changes

This section records the changes brought to the current document since its creation.

Date	Version	Description
2018/03/05	**1.0**	Initial draft

Bibliography

Len Bass. *Software architecture in practice*. Addison-Wesley, Upper Saddle River, NJ, 2013. ISBN 978-0-321-81573-6.

Google Play Game Services. Real-time multiplayer | play games services | google developers, a. URL https://developers.google.com/games/services/common/concepts/realtimeMultiplayer.

Google Play Game Services. Turn-based multiplayer | play games services | google developers, b. URL https://developers.google.com/games/services/common/concepts/turnbasedMultiplayer.

Wikipedia contributors. Composite pattern — wikipedia, the free encyclopedia, 2018a. URL https://en.wikipedia.org/w/index.php?title=Composite_pattern&oldid=820565383. [Online; accessed 5-March-2018].

Wikipedia contributors. Observer pattern — wikipedia, the free encyclopedia, 2018b. URL https://en.wikipedia.org/wiki/Observer_pattern. [Online; accessed 25-February-2018].

Wikipedia contributors. Template method pattern — wikipedia, the free encyclopedia, 2018c. URL https://en.wikipedia.org/w/index.php?title=Template_method_pattern&oldid=825387909. [Online; accessed 5-March-2018].