INTEGRATION AND ORCHESTRATION FOR MONOPOLY CLOUD SOLUTION

Stefan Mitev, Nino Karas, Marjan Gusev, Dragan Sahpaski

Faculty of Computer Science and Engineering, Ss. Cyril and Methodius University, Skopje, Republic of Macedonia

{stefan.mitev, nino.karas, marjan.gushev, dragan.sahpaski @finki.ukim.mk}

ABSTRACT

Growth in the field of the Internet Web Based Technologies and establishment of Cloud SaaS (Software as a Service) solutions, initiates various challenges and creative implementations such as solving problems of integration and orchestration of different web services. The aim of this article is to present how we have faced practical problems of integration and orchestration, realizing the game Monopoly by web based technologies. We have made several experiments to define the technology that will allow us to exchange information, without the need of global synchronization, and yet enable on-line presence by push technologies. Web sockets and web services were the most promising and appropriate technologies for our project. In this paper we present the software architecture of the solution and describe the details of integration and orchestration. The goal of the game Monopoly is to enable a platform where various cities and landmarks will be built by industries realized as web services by different vendors (students in our example) and enable on-line execution realized as Internet based game.

I. INTRODUCTION

A. *Software Trends*

Fast technological growth enables us to think more freely and creatively in the world of IT. Every day, new principles and methods that bring more power to the developers' arsenal are emerging on the web. In order to stay concurrent on the market, one has to incorporate all the trendy technologies, and keep up with every change happening. As noted in his article [1], Somasegar says that Microsoft is trying to broaden the view by incorporating these new technologies in order to satisfy their customers, and stay concurrent on the market.

The most promising technologies today are mostly web centric. The cloud paradigm (the topic of this article) is taking notable position in the IT world. Namely, the idea is to spread data, information and variety of services across the cloud (Internet), and access them from everywhere. Providers of such services also aid the need of the resource-hungry clients, talking in terms of CPU power, memory, etc. Other prominent technologies numbered in Somasegar's article are: The web platform, parallel computing, proliferation of devices, agile development process, distributed development, etc.

B. *SaaS*

SaaS (Software as a Service) is one segment of the cloud paradigm that is getting more attention in the recent years. It is also referred to, as "on-demand software" because it has an architecture that distinguishes the roles of both the provider and the client [2]. The software as a whole (physically) is owned only by the provider, while, it stands in the service of the clients.

The key players in the field of software saw this as an opportunity for even bigger profits and market share. They devised plans to integrate it in their production processes and offer their software as a service on the cloud. In the early stages, the web hosts were well suited for SaaS [3]. Today we can see many software products that are brought to us as a SaaS by the big software companies like: Google, Microsoft, etc. Lately, most MMORPG games are SaaS alike since they charge monthly fee. Some of them use micro-transactional model, which again can be classified as SaaS.

C. *Problem description*

The passion for technology led us to use cloud concepts and all technologies that support it. However developing a SaaS solution needs comprehensive understanding of the web and its technologies in order to get competitive in the IT world. So, we have started to experiment with available technologies to feel the true power beyond the theory.

The purpose of this paper is to explain integration and orchestration aspects of the solution we have realized. The initial idea was to create SaaS solution of the game Monopoly by various vendors (in our case students) that would create modules which integrated create the game flow.

Developing the game Monopoly was quite a challenge because we tried to involve as many students as possible. The game as whole was logically divided to modules that eventually later on would represent sub-projects for the students. Our approach to this realization was to create game that is SaaS alike but modularized and spread across the web, i.e. decentralized. This way we tried to create platform that is easy changeable and upgradeable.

Each module (a constituent part of the game) acts as provider of SaaS solution for the core of the game. The core, potentially integrates all the modules by using their services, and run the whole game logic. Details of implementation are defined and explained further on.

II. PROBLEM DEFINITION

We started with the concepts of the cloud and tried to find the most appropriate methods that will apply to our solution. The challenge to create the game Monopoly helped us review many ways to approach the problem at hand.

One of the first questions that arose dealt with the specificity of the technology that we wanted to use. We had to

©2012 Faculty of Computer Science and Engineering

pick a technology for the core and all the other game asset services. According to the cloud concepts of the solution, we had to plan ahead future expenditure with new submodules, and create some general concepts onto which this platform would be built with great scalability.

A. *Working environment*

Specification and setup of a working environment is required prior to the design and implementation phase. So, at the beginning we had discussion about the tools that are to be used, the way that all the communication is to be done, and other relevant matter.

Since a lot of students were working on the project, we had to have great control over the code and documentation, so to keep consistent state for everyone involved. Before the mere start, each student had to design some kind of preliminary application programming interface and provide it to the rest of his students so they can use it whenever needed. Also, during the project every student had to maintain a graphical user interface that would be used as tool for easier debugging, and controlling the work of the student to a given point in time.

B. *Architectural design*

The model of the software architecture was designed closely related to the web technologies we were about to use. Cloud concepts used in the software architecture model are presented in Fig.1.

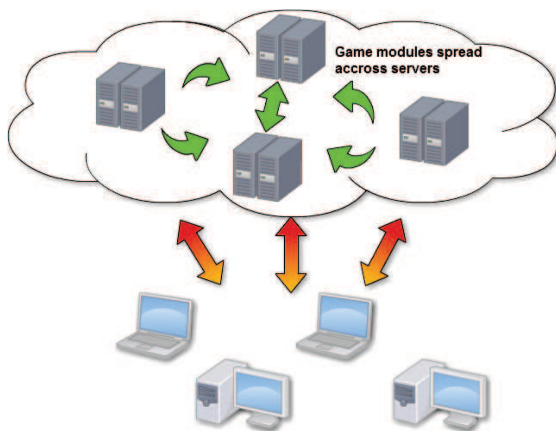

Fig. 1. Cloud concept for games.

A proper description of the architecture in terms of web sockets, web services and other commonly used web technologies is also required to realize the solution correctly.

Despite the fact that we were trying to create modularized and a loosely coupled solution, we still needed to create relationship in terms of the services that consist the initial version of the game. Using this approach enables building services that were about to be developed and integrated in this project on top of the base. The game had to be divided in two major submodules, one for the server side which would run the game flow and its logic, and other for the players' side which is actually the interface that the player uses to interact with the other players and compete in-game. The server side was further divided onto smaller submodules.

C. *Functional requirements*

The server side should contain the core module that has to integrate all the remaining modules, and run the entire game. There are some rules the Monopoly game follows [4], and those rules should be coded in the logic of the core module. Every player, prior to any further action has to role a dice. The dice number is the number of fields he should move on the board, after which, appropriate actions related to his standing-field are available. The role-dice turns are in circular manner, so, every player is invited to role the dice and continue with further actions in limited time span. If the player chooses to skip his turn, the next player is invited to roll.

The bank keeps account balances and logs for every player. When the player wants to buy or sell something, he consults the bank to get these actions done. This kind of services are integrated by the core and every time the player initiates any of the available actions, his calls are propagated to the server where the core module resides. In order to create transparency for every player, the core sends messages for the game state after the active player has performed any of the actions.

When a new game is started, the server prepares the gaming environment by loading every module in place, setting up the starting balance for every player, and starting the global timers. These and some more issues are described in detail later on.

The client side of the game should be separated on two major sub-tasks, one is the GUI, which is the look and feel of the game, and the other is code-behind, which keeps set of functionalities and Meta data for the particular player. This data should be used to keep track of some important information that will help the server to distinguish the players.

Some of the functionalities that would reside on the players' side are auxiliary and would ease the communication with the core module on the server. While others, would control the look at feel of the game in point of time. So, the client side should be very dynamic and animated because we are modeling solution for a game. According to this, we made weak logical division of these two major modules, one is more code design oriented (auxiliary functionalities), while the other is more graphically design oriented (GUI).

D. *Problem of linking different modules*

With the server and client major modules aside, we had to devise plans for linking both to work in cohesion. The server side has only few tasks that are independent of the client’s. The roll-dice invitation and some messages are those tasks the server can handle without previous synchronization with the client.

However, between invitations the server has to know the state of the active player i.e. if the player rolled and performed action or skipped his turn. This problem should be

solved by coding logic on both sides and somehow connect those through the web.

The goal is to minimize this synchronization problem and focus more on the other tasks, but consistent state across the players' browsers was crucial and tightly connected to synchronization.

III. Problem Solution

Due to variety of students working on this project, a shared project was created to reside in repository on the web. Google share looked best for these needs. We created directories and set-up the basic structure. The license type for the project was GNU GPL version 3. Next a forum was established as a place where all students can write the problems they face, so the others can help if possible. In the planning phase due to the nature of the game Monopoly it was roughly divided in logical units that later on would eventually become modules.

Using this approach, we could create subprojects and divide the work among the students. Then, each student had to describe his project in terms of brief abstract description and a must-have application programming interface (API). The API would help the others understand the services one's project offers, but also help for easier integration with the rest of the modules. After initial adjustments, we had to pick the technologies that will help us realize the game. The main technology used was ASP.NET along with JavaScript, CSS, etc. Despite the small size of the project we still used database for the game's data, in this case the MS SQL Server. To connect the core module and the players we used service Pusher, which helped us a lot in this project. All the modules were mainly created as web services.

After this initial set-up we were ready to start designing the modules and the connections among them. Reference diagrams were created to illustrate the modules and their connections. This would help us overview the system of modules as a whole, and derive the architecture of the game.

A. Architectural model

During the planning phase we had rough overview of the modules and their connections in the system. In the design phase we focused more on this issue and started to build more complex and detailed software architecture design, which would address all the sub-problems that would arise. At the highest level, our design consisted of many modules that would potentially reside on different servers across the web.

All of their services would be exposed to one core module that would run the whole game flow, as presented in Fig.2. Again, this core module could reside on yet another server. This way we defined loosely coupled modularized game that would be easier to maintain and upgrade.

The client's (player) side could be again seen as module, but this module was unique in the way that the core module controlled its characteristics. In other words, the client's module is capable of doing specific actions if allowed by the core module. This is the way we control the actions of the players. When one gets the chance to play, the others are limited in their actions and wait for their turn to come. The client's module is dynamically created and given to the client upon his request.

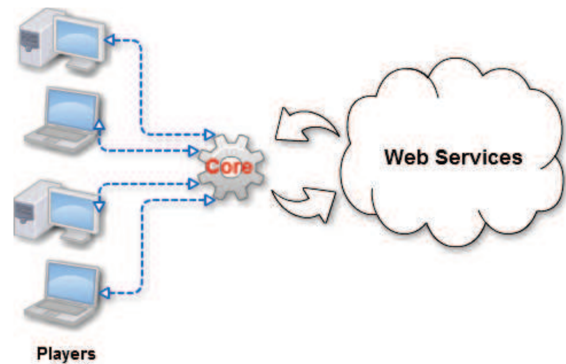

Fig. 2. Architecture of the Monopoly cloud solution.

All of the modularization and interconnection among the modules is due the fact that we tried to create cloud solution for the game. All modules would act as SaaS structures and the core module, which also integrates them, would use their services.

B. Server side

The server side consists of the core module and all the rest of the modules that are auxiliary and help the core in creating the game flow as presented in Fig.3.

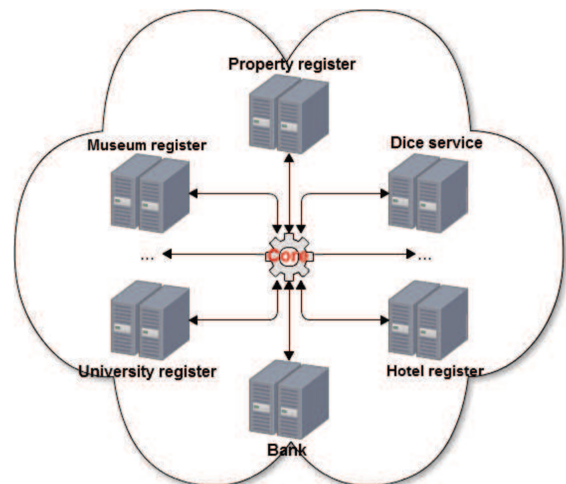

Fig. 3. Server side modularization.

The core is module where the game logic is written. It runs a loop that doesn't stop until the end of the game. This loop repeatedly invites the players to play their turns. The players are invited in circular manner. If the active player doesn't want to roll the dice (the first step of one's turn) his turn is skipped and the next player is active. After the player rolls the dice, the server dynamically creates the client's module and returns it to the player, after which the player is able to proceed with further actions during his turn. Using this approach the core refines the client module upon the player's actions, i.e. it acts as an observer.

Some problems occurred due to the characteristics a web application possesses. In a web application the memory allocation is lost after the server processes the request and returns the result. This was unacceptable for us because there were some data that were needed constantly at the client's side.

These data had to be kept consistent across all the players, and not lost upon one's call. Luckily we had our core defined to run in a loop that meant that its life cycle was as long as the game lasted. So, we could just put all the data in the core and let the clients use it. But, this turned out to be hard to achieve because another problem occurred all of a sudden. The core couldn't just run in loop till the game is not finished since it would slow down the client's requests by introducing delay in the response. To fix this issue we decided to use threading and separate the loop of the game in separate thread that would run in the background with the server's system clock. The client's calls would be asynchronously handled and their response should be returned by the core. But again one problem showed up after the separation of the core in another thread. A process put to work in different thread, is locked for outside communication and could not be accessed. It can only emit messages and calls to the outside. This was again unacceptable for us because we kept the data inside the core and the clients had to use it. After a while, we put an end to this chain of problems with introduction of the "mediator object" which solved these issues in elegant way.

C. *Mediator object*

The mediator object helped us to allow communication between the core and the players in both ways as presented in Fig.4.

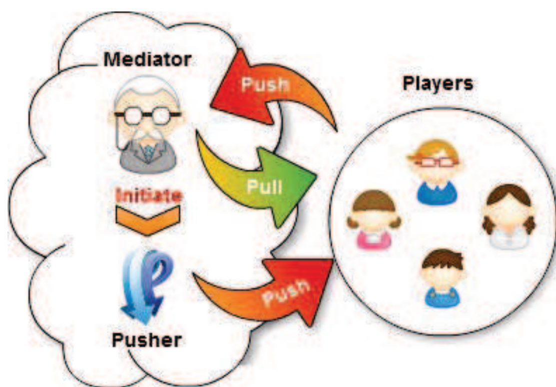

Fig. 4. Mediator object illustration.

Because we had the core running in loop we could use an object inside it and grant access to all the players and the core itself. This object was created as a singleton, and whenever some change happened everyone could see it. When the players need to read or write data in the object, they just need to get the instance of the object through its class and do the desired changes. Next time the core checks the state of the object, it would see the changes made by the last player who altered its state. So, this object was mediator between the players and the core, and set between both in the cloud. The data inside the object was separated in two groups. The first group consists of data that is not changed at all since the start of the game, while, the other group consists of data that is changed upon player turn changes.

So, this object should have partially changeable context that is changed on regular time intervals. The data that is not changed consists of instances of the other auxiliary modules that are needed by the core. The changeable data is consisted of the information for the active player. Those are primarily: id, name and field index of the active player. These data –as shown later in the problem of linking – help the core to distinguish the players and their allowable actions. After his actions, the player changes the context of the object so the core can see these actions, and based on them refine the client's module. Every player has a time span to perform his actions during his turn. When this time expires, the core switches context of the mediator object and prepares it for the next player, after which invites him to roll the dice.

D. *Client side*

The client side is described by the client's module, which actually is the served page by the core. This page contains various functionalities (JavaScript) enabled or disabled by the core before it serves the page to the player upon his actions. For reasons explained later, inside this page there is a hidden field that keeps the id of the player.

There are several functionalities that are independent of the core and help with dynamically decorating the GUI. Other functionalities are indirectly dependent of the core. Their use is the propagation of the player's requests to the core with the help of the mediator object. Some of the functionalities are there to assist the Pusher service which can invoke them if instructed by the core. These functionalities provide the core with the mechanisms to dynamically change the GUI of the player when needed. One such example is the message stream that the core is producing. When the active player performs an action, the core notifies all the other players with help of the Pusher service, by instructing it to invoke the functionalities for writing a message in the clients' browsers.

Some of the actions the player can perform are those that determine the possible moves he can make in the game Monopoly. The actions can be divided in two groups, the first of which are those that can be initiated by the player, and the other is those that are performed upon specific event happens. Such actions are buying or selling houses, hotels, fields or other properties. The one that are event based could be: stepping the finishing line, stepping on a field that is owned by another player, going to jail, chance cards, community chest, etc. The rules of the game are coded inside the logic of the auxiliary services. These rules limit the player in terms of his actions.

E. *Game assets*

The auxiliary modules create the gaming environment. These modules are: bank, property register, construction-companies register, wineries register, hotels register, museums register, universities register, chance cards, community chests, etc. These modules are independent of the core but they offer their

services to the core and are of crucial importance for the game. The bank keeps track of the players' accounts, their balance and transaction logs. It governs rules for withdrawing or depositing money. It takes properties under mortgage and buys or sells them if needed. Before the game begins, the bank gives every player the same starting amount of money.

The property register keeps track of the properties' owners. It governs the sell or buy price for every property, the taxes for building hotels and houses, and other property related matters. It also follows rules for building hotels or houses on the actual field. For example, the player needs to own some quantity of houses before building hotels. The fields are divided into groups with unique color. When the player wants to build house he has to own all the fields of the same group, or otherwise he is not allowed to build any house. The property module governs these and some more rules.

The other register-like modules help in decorating the game for better gaming experience. When the active player is located on any of the fields, these modules give information about the objects and landmarks located on that field. So, the player can also use the game as an educational tool and learn about the many objects and landmarks located in a specific region (in our case is Macedonia).

F. Linking the modules

With all the modules designed and implemented, it was time to link all of them and run the game. We wanted to care less about synchronizing everything the hard-coded way. To avoid the synchronization we found great service that helped us with the linkage problem. The service Pusher is using various technologies to enable communication through the web on great scale. Its main technology is web sockets, which enables full-duplex communication between a server and its users, as depicted in Fig.5.

Fig. 5. Pusher service illustration.

The service uses the concept of broadcast channels and allows the server to emit messages through this channel to all the listeners attached to it. So, in our case the players are listeners and the core is the emitter. This creates two-way observer pattern, as earlier mentioned, the core listens for changes made by the player with the help of the mediator object, and the players listen to the changes made by the core, again with the help of the mediator object. Since the pusher service is broadcasting the messages and changes the context of the client's module according to the active player, we had to use the ids of the hidden field at the client and that of the mediator object to separate the actions for the active player and those for the other players. When a match occurs, the actions for buying and selling are unlocked for that player. The other players just receive the notification messages during this period of the game.

Prior to the game start all the players are attached to the global channel where the core emits the messages and runs the whole game. The core can initiate call to one of the functionalities that reside on the client's side with help of the Pusher service. This way it can divide the need of resources and process some of the work on the client's side.

IV. Game flow

The game starts with creating names and waiting for the others to be ready for start. When everyone is ready, a timer is counting down after witch every player is redirected to the main game screen (the board). The first player is given the chance to roll the dice and a timer counts from 10 to 0. If the player doesn't roll during this time interval, his turn is passed and the next player is invited to roll the dice. If the player rolls, the number he rolled is the number of steps he should move on the board.

When the player lands on a field, the various register-like services provide information about the objects and landmarks residing on that field. The player can choose to visit some of them and gain some points. For the visit he is charged small amount of money. If the field is not owned by anybody, the player is able to buy it and later on build houses and hotels on it. If the field is owned (a privet property), the player is charged by a fee or tax for passing by on foreign field.

There are some fields that can only by bought but there's no option for building anything on them. These fields are the railway stations, electricity factory, and similar. But again, if foreigner steps on these fields, he is charged for passing by. The chance and community fields are special ones. When the player lands on those, he can get some bonuses in terms of free roll or money.

The "police" field sends the lander to the jail where he is put for the next two turns. If there is lander on the jail field without previously sent from the police, then he is just visitor and nothing is done to him.

The rules for building houses and hotels are as follows: The player can build house if he owns the fields in the same group as the actual one. Hotels can be built if the player owns at least one house on all the fields in the same group as the actual one. If the player needs money to raise his budget, then he can put some of his properties on mortgage at the bank. These are some of the rules that the game follows, and more about them can be read on the previously given reference.

V. Comparison

There are many Monopoly games on the Internet [6, 7]. Some of them run using adobe flash technology, others as

embedded programs (java servlets). No matter the technology of implementation, all the games we've found so far share the same characteristic by being implemented as one module. These games are indivisible and usually reside on one server. Our solution exposes the cloud SaaS structure and creates platform that could be used by many types of games. Using this approach it is easier to upgrades and maintain the game.

The modules are loosely coupled and are only bound by the core which is yet another module. This scalability allows adding new modules that would decorate the game even further.

Also, the reliability is on higher level since the modules are spread across many servers. If one of the modules is not working the game could possibly continue without a loss or any notable damage, on the other hand, games that are tightly connected tend to be more erroneous because it is harder to reveal the bugs. If a system failure occurs then the whole game would not run.

It is also harder to add new changes to existing parts of the game because sometimes this would require recoding or even redesigning the old parts. In a modularized way, the module that is to be changed can be isolated and worked on its own, or even replaced with new one.

Another issue that is good to compare is the fact that these types of games use the resources of the clients exclusively. Whereas the cloud SaaS one uses better share of the resources available to the server, and less of the client's resources.

VI. Conclusion

We present a cloud solution of a Monopoly game with special intention how we solved the integration and orchestration problems using appropriate web technologies. Brief overview was taken on the software trends about using the cloud and developing a SaaS solution.

Along the phase of design and development of the project we faced some obstacles that we succeeded to overcome. A sophisticated software engineering approach was used during the project, starting wit the process of planning, design and implementation and roll-out by managing different teams and corresponding background using various of technologies.

According to Gartner [8], the cloud and PaaS (Platform as a service) is about to dominate the field of IT in the following years. We are surrounded by the globalization and virtual socialization, and this leads us to think for the cloud, examine it well and learn more about it. The cloud itself could minimize the software piracy that many developers struggle with. Since the software is on the cloud, no one except the developer has the exclusive right to run it and manipulate with it. The users of the services that this software offers are not aware of the type or the way the software behaves. They just hand off the parameters and get the answers. This changes the business model from one-time charge, to per-service or monthly fee charge, since the users pay only for the service and not for the software or its license. Additionally, advertising and marketing could be promoted through the cloud, which again adds more benefits and stand as financial source for the provider. Many new companies that need services from software that would cost many thousands could leverage on the providers of a cloud solutions for affordable monthly fee. Productivity is not limited by any means, since there is great support from these providers that have the resources and power to aid the needs of the startups, or even serious business players. The providers of the cloud solutions care about their software maintenance and upgrade it constantly, which means that the users would not need to worry about these issues. Many businesses that run traditional on-premise in-house software would cut off the costs for such maintenance, if they switch to cloud solutions. The cloud changes the way users interact with the software, the way the businesses gather and process information, or keep data [9]. It promotes the globalization in the IT world more than ever.

References

[1] S. Somasegar, "Key Software Development Trends", *Microsoft Developer Division Somasegar's blog*, 23 Feb 2010, blogs.msdn.com/b/somasegar/archive/2010/02/23/key-software-development-trends.aspx

[2] *Software as a Service, Cloud taxonomy*, 2012, online, cloudtaxonomy.opencrowd.com/taxonomy/software-as-a-service/

[3] J. Brodkin, "Trends in Software as a Service", *PC World,* 16 Jul 2007 www.pcworld.com/article/135119/trends_in_software_as_a_service.html

[4] *Official Monopoly game rules*, Tripod, online, richard_wilding.tripod.com/monorules.htm

[5] *Pusher software product*, online, http://pusher.com/

[6] *Monopoly game*, online, www.hasbro.com/monopoly/en_US/

[7] *Play Monopoly Online*, Pogo com, online, board-games.pogo.com/games/monopoly

[8] D. Roe, "Gartner's 5 trends for enterprise software", 4 Feb 2011, online, www.cmswire.com/cms/enterprise-cms/gartners-5-trends-for-enterprise-software-010089.php

[9] Gartner news, "Gartner outlines five cloud computing trends that will affect cloud strategy through 2015", online, www.gartner.com/it/page.jsp?id=1971515