Uniwersytet Warszawski
Wydział Matematyki, Informatyki i Mechaniki

Paweł Bedyński
Nr albumu: 239764

HEPI – wielojęzykowy komunikator internetowy w świecie 3D

Praca licencjacka
na kierunku INFORMATYKA

Praca wykonana pod kierunkiem
dra Aleksego Schuberta
Instytut Informatyki

Maj 2009

Oświadczenie kierującego pracą

Potwierdzam, że niniejsza praca została przygotowana pod moim kierunkiem i kwalifikuje się do przedstawienia jej w postępowaniu o nadanie tytułu zawodowego.

Data Podpis kierującego pracą

Oświadczenie autora (autorów) pracy

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam również, że przedstawiona praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w wyższej uczelni.

Oświadczam ponadto, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną.

Data Podpis autora (autorów) pracy

Streszczenie

W pracy znajduje się opis projektu wykonanego w ramach Zespołowego Projektu Programistycznego. Istotą projektu jest stworzenie wirtualnego świata, w którym każdy użytkownik opiekuje się swoją postacią. Celem prac było symulowanie rzeczywistości oraz umożliwienie użytkownikom z różnych kultur swobodnego sterowania swoją postacią, włączając w to możliwość komunikowania się z innymi użytkownikami. Poprzez zastosowanie prostych w obsłudze, nowoczesnych i intuicyjnych rozwiązań technologicznych udało się uprościć komunikację poprzez połączenie aplikacji typu *gadu–gadu* z trójwymiarowym generowaniem świata. Dzięki prezentowanej aplikacji spotkania użytkowników nie będą ograniczały się do „rozmowy" tekstowej. Nowością jest to, że można zobaczyć wirtualną projekcję rozmówcy generowaną zgodnie z jego osobistymi preferencjami.

Słowa kluczowe

komunikator, świat 3D, renderowanie, OGRE, aplikacja client–server, tłumacz

Dziedzina pracy (kody wg programu Socrates-Erasmus)

11.3 Informatyka

Klasyfikacja tematyczna

D. Software
D.2. Software engineering
K.6.3. Software Management

Tytuł pracy w języku angielskim

HEPI – multilanguage internet communicator in 3D world

Spis treści

Wprowadzenie

Aplikacja HEPI służy do komunikacji pomiędzy użytkownikami mówiącymi w różnych językach. W celu uatrakcyjnienia konwersacji każdy dostaje trójwymiarową postać i mieszkanie w wirtualnym świecie. Dzięki temu rozmowy mają być trochę bliższe tym, które prowadzimy w rzeczywistym świecie.

Projekt podzielony jest na pięć głównych części:

- *Aplikacja klienta*
- *Konfigurator*
- *Serwer*
- *Aplikacja administratora*
- *Baza danych*

Aplikacja klienta wraz z *Konfiguratorem* to programy, które użytkownicy instalują na swoim komputerze. Ich zadaniem jest rejestracja i konfiguracja postaci oraz wyświetlanie grafiki trójwymiarowej i prowadzonych rozmów. Warto zwrócić uwagę na oddzielenie od siebie tych dwóch funkcjonalności (konfiguracji i wyświetlania grafiki). Szczegóły dot. *Konfiguratora* znajdują się w rozdziale 2.2, a *Aplikację klienta* omawiamy dokładniej w rozdziale 2.1.

Kluczową rolę w komunikacji odgrywa aplikacja *Serwera*. Łączy on użytkowników ze sobą przesyłając pomiędzy nimi komunikaty. Np. gdy użytkownik idzie do jakiejś nowej pozycji to informuje o tym serwer, a ten przesyła informację dalej do wszystkich zalogowanych użytkowników.
Serwer pobiera dane z *Bazy danych* (szczegóły w rozdziale 2.5). Odpowiada też za tłumaczenie wiadomości użytkowników, kontaktując się z translatorem Google©.
Przez tę aplikację odbywa się również cały proces rejestracji i konfiguracji postaci. Szczegóły w rozdziale 2.4.

Do sprawnego zarządzania wirtualnym światem przygotownana została *Aplikacja administratora*. Służy ona do dodawania nowych obiektów na mapie świata (drzewa, fontanny itp). Dzięki niej można łatwo dodać nowe postaci i modele. Umożliwia również usuwanie i banowanie użytkowników, którzy nie stosują się do regulaminu. Aplikacja omówiona jest w rozdziale 2.3.

Oprócz tego powstały drobniejsze projekty takie jak strona WWW, *instalator* programu oraz pliki pomocy.

Rozdział 1

Założenia

1.1. Definicje

Podstawowe części projektu i ich nazewnictwo:

- *Aplikacja klienta (C++)* – aplikacja wyświetlająca świat 3D. Dostarczana użytkownikowi wewnątrz *instalatora*.
- *Konfigurator (C#)* – aplikacja konfigurująca *Aplikacje klienta*. Umożliwia przeprowadzenie pełnego procesu rejestracji nowego użytkownika. Dostarczana użytkownikowi wewnątrz instalatora.
- *Serwer (C#)* – aplikacja *Serwera* łącząca poszczególne warstwy. W ramach projektu Serwer działa na wykupionej przestrzeni dyskowej na terenie USA.
- *Aplikacja administratora (C#)* – aplikacja służąca do administrowania zasobami bazy danych. Szczegółowy opis interakcji z innymi warstwami projektu został opisany w rozdziale 2.3.
- *Baza danych (MS SQL)* – baza danych Microsoft© SQL założona na serwerze wydziałowym *terminator.mimuw.edu.pl*.
- *Strona WWW (ASP.NET)* – aplikacja umożliwiająca użytkownikom pobranie programu i przeczytanie krótkich informacji o projekcie.
- *Instalator (inno setup)* – aplikacja instalująca klienta na komputerze użytkownika.

Inne pojęcia występujące w dokumencie i ich znaczenia:

- **Interfejs** – część programu odpowiedzialna za interakcje z użytkownikiem.
- **Login** – nazwa użytkownika zawierająca jedynie znaki [a–Z,0–9]*. Minimalnie 6 znaków, maksymalnie 32.
- **Socket** – końcowy węzeł komunikacji dwukierunkowej w sieci opartej na protokole IP, takiej jak np. Internet.
- **Trigger** – procedura wykonywana automatycznie jako reakcja na pewne zdarzenia w tabeli bazy danych.

- **MVC** – model View Controller, wzorzec projektowy którego głównym założeniem jest wyodrębnienie trzech podstawowych komponentów aplikacji: Modelu danych, interfejsu użytkownika, logiki sterowania,
- **WWW** – World Wide Web.

Wyjaśnienia pojęć dotyczących wyświetlania świata trójwymiarowego:

- **Engine** – zestaw bibliotek umożliwiający wyświetlanie grafiki w czasie rzeczywistym.
- **Model** – wektorowy zapis kształtu trójwymiarowego. Klasa obiektów trójwymiarowych.
- **Obiekt** – wszystko co jest postawione na mapie (drzewa, budynki, fontanny, itd). Instancja modelu. Każdy model obiektu może występować w kilku miejscach.
- **Budynek** – mieszkanie użytkownika. Podgrupa zbioru obiektów.
- **Model postaci / Awatara** – wektorowy zapis kształtu postaci.
- **Skin (skórka)** – tekstura nałożona na wektorowy obiekt.
- **Szkielet postaci** – zbiór odcinków symulujących kości. Umożliwia łatwe animowanie postaci.
- **Animacja postaci** – opis poruszania się kości.
- **Promień słyszalności** – komunikat tekstowy wysyłany przez użytkownika dociera tylko do użytkowników znajdujących się wewnątrz kuli o środku w punkcie aktualnej pozycji nadawcy, a promieniu równym z góry ustalonemu *promieniowi słyszalności*.

1.2. Podstawowa funkcjonalność

1.2.1. Od strony użytkownika

Możliwości użytkownika dostępne przez *Konfigurator*:

- rejestracja w systemie,
- utworzenia własnego budynku na mapie,
- zarządzanie profilami użytkowników korzystających z aplikacji.

Możliwości użytkownika dostępne przez *Aplikacje klienta*:

- Sterowanie własną postacią w wyświetlanym przez *Aplikacje klienta* trójwymiarowym świecie, w którym obecne będą również awatary należące do innych ludzi. Postać w grze potrafi się poruszać oraz wykonywać proste czynności,
- Komunikacja z innymi użytkownikami (w formie tekstowej) za pośrednictwem awatarów, jak również rozmowy między wieloma postaciami jednocześnie pod warunkiem, że znajdują się dostatecznie blisko (patrz – *promień słyszalności*),
- Komunikacja użytkowników jest tłumaczona zgodnie z indywidualnymi – ustalonymi podczas rejestracji – preferencjami językowymi,

- Symulacja odwzorowuje świat rzeczywisty zarówno poprzez implementacje świata wewnętrznego (fizyka świata rzeczywistego, sam wygląd świata), jak i poprzez interakcje użytkownika ze światem (zapamiętywanie pozycji użytkownika, tak by po powtórnym zalogowaniu wrócił w prawie to samo miejsce).

1.2.2. Od strony administratora

Możliwości administratora systemu dostępne przez *Aplikacje administratora*:

- zarządzanie użytkownikami systemu,
- edycja mapy,
- dodawanie nowych awatarów (modeli i skórek),
- dodawanie nowych obiektów (modeli i skórek),
- przeglądanie/wyszukiwanie informacji w naszym systemie.

Rozdział 2

Opis części systemu, zastosowanych technologii i rozwiązanych problemów

2.1. Aplikacja Klienta

Aplikacja klienta – podzielona na 3 współpracujące ze sobą warstwy – jest główną częścią całego projektu. Koncepcja architektury aplikacji jest zbliżona do modelu *MVC*.

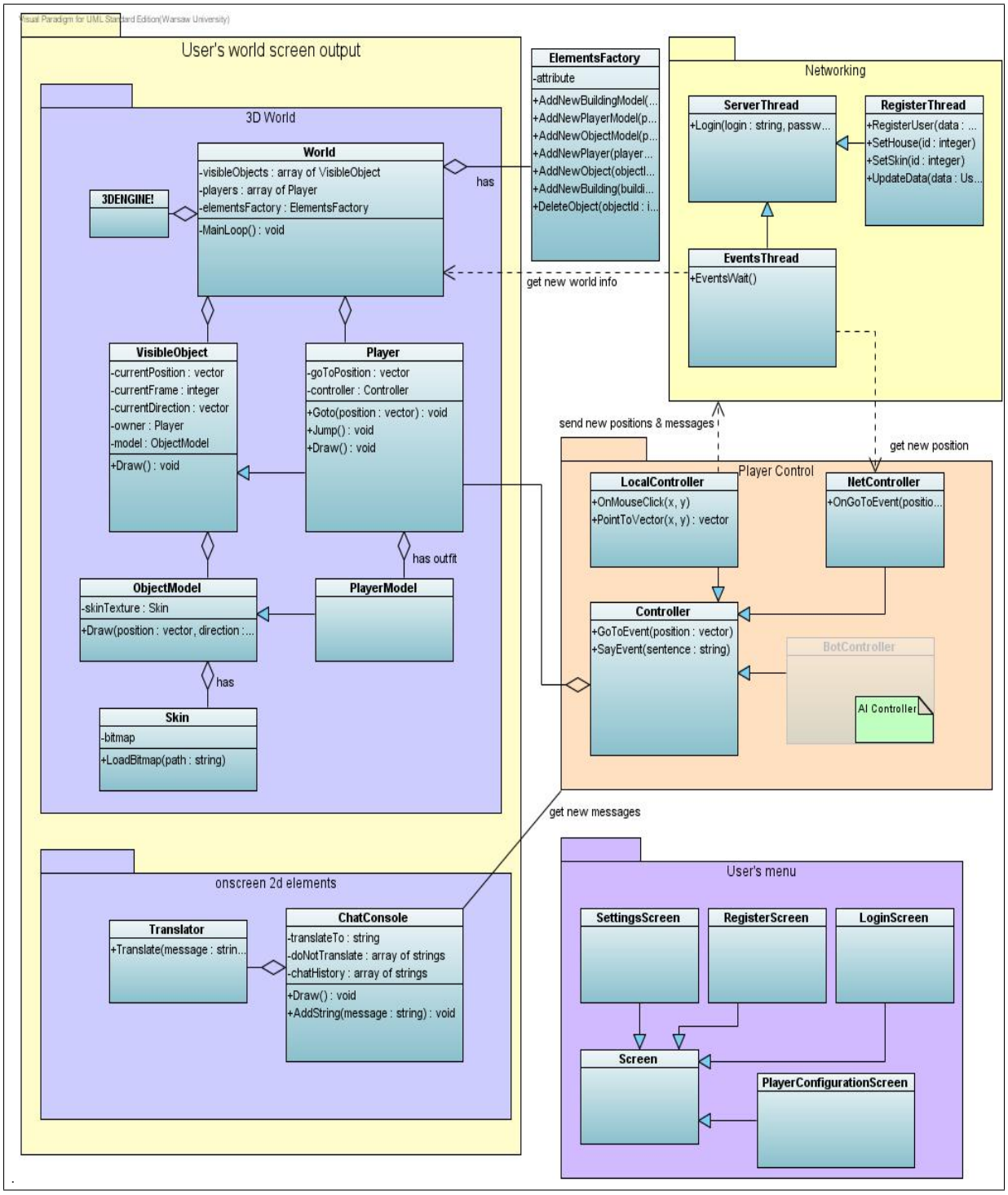
2.1.1. Ogre (View)

Podstawą *Aplikacji klienta*, jest silnik odpowiedzialny za wyświetlanie grafiki. W projekcie wykorzystano technologię OGRE ze względu na:

- kompatybilność z .NET,
- prężnie działającą społeczność open–source,
- szeroką gamę efektów graficznych (deszcz, mgła, ogień itp.),
- technologia była wykorzystana już w kilku popularnych komercyjnych produkcjach.

Silnik *Ogre* ma budowę obiektową. Podstawowym obiektem jest `Root`, który odpowiada za tworzenie innych obiektów. Za wyświetlanie odpowiada `SceneManager`. Każdy obiekt trójwymiarowy jest węzłem w drzewie wszystkich obiektów. Z każdym węzłem powiązany jest jego model trójwymiarowy. Szczegóły można znaleźć w [1] i [2].

Każdy obiekt świata umie "wyświetlić się" za pomocą funkcji niższego poziomu, które w chwili obecnej wykorzystują silnik *Ogre*, ale zmiana technologii jest możliwa bez przepisywania całości kodu. Takie podejście ułatwiło nieco testowanie poszczególnych warstw *Aplikacji klienta*.



Rysunek 2.1: Diagram klas *Aplikacji klienta*

2.1.2. Dane z serwera (Model)

Po skonfigurowaniu programu, *Aplikacja klienta* otrzymuje (od *Konfiguratora*) login i hasło. Po udanym zalogowaniu tworzony jest wątek zdarzeń `EventsThread` i rozpoczyna się wyświetlanie świata.
Od tego momentu wszystkie nowe dane prezentowane w programie pochodzą z komunikacji z *serwerem*.

Socket połączenia z *Serwerem* po otrzymaniu zdarzenia działa następująco:

- jeśli zdarzenie dotyczy wyświetlania świata (nowy budynek, nowa postać, usunięcie obiektu, etc.), to uruchamia odpowiednią funkcję fabryki elementów w klasie `World`,
- jeśli zdarzenie dotyczy kontrolowania którejś z postaci użytkownika (np. „idź do punktu p", „powiedz 'ala' "), to uruchamiana jest funkcja odpowiedniego kontrolera postaci.

2.1.3. Kontroler (Controller)

Główną koncepcją projektową było podłączenie do każdej postaci osobnego kontrolera. W rezultacie postacie sterowane przez innych graczy mają podłączony kontroler oparty o protokół, a gracz lokalny ma kontroler związany z myszką i klawiaturą.
Motywacją był fakt, że dla obiektu trójwymiarowego w świecie nie ma różnicy, czy o jego przemieszczeniu się zadecydował użytkownik, czy może dostał rozkaz z *Serwera*. Ważne jest, że otrzymał komunikat "idź do punktu X,Y,Z" i ten komunikat realizuje.

2.1.4. Rozmowy

Użytkownik wysyła i odbiera wiadomości tekstowe za pomocą okna rozmowy – dolnej części głównego okna.
Rozmowy przychodzą w jednej lub dwóch wersjach językowych. Jeśli język używany przez rozmówcę jest inny od języka używanego lokalnie to wiadomość jest automatycznie tłumaczona. Przesyłana jest również wersja oryginalna - dzięki temu, jeśli użytkownik choć trochę zna język rozmówcy, to w przypadku ewentualnych błędów tłumaczenia użytkownik ma szanse sprawdzić poprawność tłumaczenia.

2.1.5. Diagram klasy

Na rysunku 2.1.1 zaprezentowano uproszczony schemat klas *Aplikacji klienta*. W związku z angielskimi nazwami zmiennych i klas, dla zwiększenia czytelności diagram umieszczono w całości po angielsku.
Budowa została w dużym stopniu zdeterminowana zarówno przez protokół, jak i przez silnik graficzny.
Za "Widok" można uznać lewą stronę tego diagramu, kontrolery przedstawione są po prawej stronie w różowym prostokącie, a w prawym górnym rogu znajdują się klasy odpowiedzialne za odbiór danych z serwera, czyli nasz model.

2.1.6. Struktura programu na dysku twardym

W poniższym schemacie nie uwzględniono bibliotek i plików konfiguracyjnych wykorzystywanych w aplikacji (Ogre, MOGRE, .NET).

```
{Prog}\Hepi.exe              // główny plik wykonywalny
{Prog}\HepiConfigurator.exe  // główny plik konfigurator
{Prog}\hepi.cfg              // plik konfiguracyjny
{Prog}\textures\             // pliki z grafiką statyczną
{Prog}\Avatars\              // modele postaci
{Prog}\AvatarsSkins\         // tekstury postaci
{Prog}\Objects\              // modele obiektów
{Prog}\ObjectsSkins\         // tekstury obiektów
```

2.1.7. Formaty danych

Formaty są narzucone przez *engine*. W miejscach gdzie był wybór, został on podjęty zgodnie z poniższą tabelą (w tym przypadku zawsze motywacją była prostota implementacji):

Rodzaj obiektu	**Format pliku**
Modele postaci	Ogre 3D (.mesh)
Skiny postaci	JPEG (.jpg)
Modele obiektów	Ogre 3D (.mesh)
Obiekty przezroczyste	PNG (.png)
Mapa	Format własny (.png)
Tekstury	JPEG (.jpg)

2.2. Konfigurator

Konfigurator jest aplikacją wstępną przed *Aplikacją klienta*. Celem *Konfiguratora* jest w praktyce przygotowanie środowiska pod działanie *Aplikacji klienta* tj. wybór postaci (loginu i hasła), którą użytkownik chce się zalogować. Przygotowywana jest również lista używanych w danej sesji *Serwera* plików.
Aplikacja *Konfiguratora* ma budowę przypominającą automat stanowy *(rysunek 2.2)*

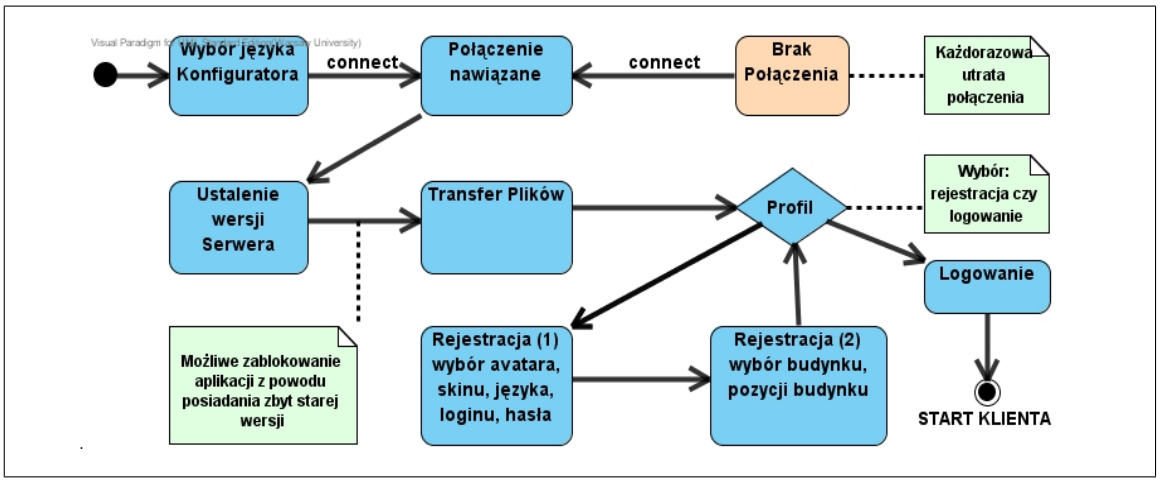
W *Konfiguratorze* – podobnie jak w pozostałych częściach projektu – większość implementacji oparta jest na strukturze modułowej. Wykorzystywane moduły w kolejności chronologicznej (zgodnie z przebiegiem konfiguracji) to:

1. Moduł *FileTransfer* – odpowiedzialny za nawiązanie połączenia z serwerem, a następnie synchronizację danych oraz utworzenie listy plików dostępnych dla *Aplikacji Klienta*. Należy zwrócić uwagę, że nawiązanie połączenia z *Serwerem* jest nierozerwalnie związane z przeprowadzeniem procesu synchronizacji.

2. Moduł *Packer* – wypakowuje pobrane pliki do plików akceptowalnych dla *aplikacji Klienta*

3. Moduł *XML* – zarządza profilami graczy, umożliwia zdefiniowanie profilu domyślnego, który zapisuje się w pliku XML–owym.

Struktura implementacji graficznej *Konfiguratora* składa się z czterech części: pierwszą jest obszar przycisków kontrolujących przechodzenie pomiędzy panelami. Drugą jest kronika *Serwera*, w której wyświetlają się informacje np. o nawiązaniu połączenia, lub raporty z prac synchronizatora danych. Obszary trzeci i czwarty to równoprawne kontenery (lewy, prawy), w których mogą być wyświetlane panele (katalog `/Panels`). Płynność grafiki podczas absorbujących operacji *Konfiguratora* (nawiązywanie połączeń, pobieranie plików) zapewnia mechanizm `BackGroundWorker`.



Rysunek 2.2: *Konfigurator* – diagram stanów

2.3. Aplikacja Administratora

Aplikacja została stworzona w celu łatwej administracji systemem, bez konieczności dokładnego poznawania jego schematu działania.

2.3.1. Funkcjonalność

Główną funkcjonalnością aplikacji jest zarządzanie tzw. nową treścią (ang. new content) w naszym świecie. Chodzi o możliwość:

- Dodawania/usuwania modeli awatarów,
- Dodawania/usuwania skórek modeli awatarów,
- Dodawania/usuwania modeli obiektów,
- Dodawania/usuwania skórek modeli obiektów,
- Zmieniania planszy tj. dodawania/modyfikowania/usuwania znajdujących się na niej obiektów.

Oddzielną funkcjonalnością jest możliwość przeglądania kroniki naszego systemu, oraz zarządzania użytkownikami (banowanie/odbanowywanie na poszczególne części naszego systemu).

Aplikacja udostępnia również możliwość wyszukiwania oraz przeglądania każdego elementu naszego systemu.

2.3.2. Problemy

Głównym problemem napotkanym przy pisaniu tej aplikacji, okazała się współpraca z *Bazą danych* i *Serwerem*. Na styku tych trzech części naszego systemu pojawiły się problemy z synchronizacją i spójnością danych.
Zastosowano następujące rozwiązania:

- Wszelkie zmiany dokonywane w *Bazie danych* odbywają się poprzez atomowe procedury,

- Każda newralgiczna, modyfikująca bazę procedura, sprawdza stan *Serwera* – w skrajnym przypadku blokuje możliwość modyfikacji,
- Każde zapytanie operuje na niemodyfikowalnych widokach.

Zastosowano też zabezpieczenie przed atakiem typu SQL Injection (oznaczanie parametrów).

2.3.3. Technologie

Aplikacja ta, poza standardowymi technologiami (C# – obsługa sieci, połączenia z bazą danych etc.), wykorzystuje Windowsowe© Okienka dla stworzenia wysoce intuicyjnego interfejsu.

Główne okienka to:

- FormError – obsługa błędów,
- FormLogin – obsługa logowania do systemu,
- FormMain – główne okno z podziałem na zakładki, zapewniające większość funkcjonalości,
- FormPacker – obsługa pakowania plików do formatu .hepi.

2.3.4. Formaty danych

Rodzaj obiektu	**Format pliku**
Modele awatarów	Ogre 3D (.mesh) spakowane do .hepi
Skiny awatarów	JPEG (.jpg)
Modele obiektów	Ogre 3D (.mesh) spakowane do .hepi
Skiny obiektów	PNG (.png)
Mapa	Format własny (.png)

2.3.5. Struktura programu na dysku twardym

Aplikacja przeznaczona jest do uruchamiania w głównym katalogu *Serwera*.

```
{Prog}\Admin.exe              // główny plik wykonywalny
```

2.4. Serwer

Aplikacja *Serwera* składa się – oprócz podstawowej części omówionej później – z następujących modułów:

- *Moduł FileTransfer* – odpowiada za transfer plików modeli i skórek (awatarów i obiektów) z lokalnej pamięci *Serwera* do *Konfiguratorów*. Przy starcie *Serwera* przeprowadzany jest `Integrity Test`, w ramach którego *Serwer* dowiaduje się jakich plików „używa baza". W przypadku niepomyślnego przebiegu testu (oznacza to, że serwer nie posiada pliku, którego używa *Baza danych*) *Serwer* zostaje wstrzymany, a odpowiedni komunikat informuje o przyczynie niepowodzenia testu. Pomyślny przebieg testu gwarantuje,

że *Serwer* posiada wszystkie niezbędne pliki. Jeśli w katalogach transferowych znajdują się dodatkowe pliki, to *Serwer* o tym poinformuje, lecz nie powoduje to wstrzymania działania aplikacji. W ten sposób *Serwer* ustala obowiązująca listę plików danej sesji, której będzie wymagał od *Konfiguratorów*. Po nawiązaniu połączenia TCP między *Konfiguratorem* a *Serwerem* uzgadniana jest oficjalna lista plików. W przypadku braku któregoś pliku *Serwer* wysyła potrzebny plik.

- *Moduł Translator* – odpowiada za przeprowadzenie tłumaczenia tekstów przychodzących od *Aplikacji klienta* w komunikatach `MSG_CHAT`. Podczas logowania *Serwer* tworzy dla użytkownika obiekt klasy `User`, w którym zapisane są m.in. informacje o wybranym języku. W ten sposób po dojściu wiadomości i wywołaniu metody `WhoCanHearMe` *Serwer* konstruuje zapytanie do `serwera google translate` © , i tak tworzona jest lista wszystkich potrzebnych translacji według języków określonych przez użytkowników znajdujących się w obrębie promienia słyszalności. Odpowiednie tłumaczenia są wtedy gotowe, by rozesłać je poszczególnym odbiorcom.

- *Moduł ObjectsControl* – odpowiada za kontrole nad obiektami w *Aplikacjach klienta*. Poprzez wywoływanie komunikacji w dwie strony (do *Bazy danych* oraz do *Aplikacji klientów*) utrzymywana jest pamięć o tym, który obiekt został wysłany do którego użytkownika (tabela `ObjectTransfer` w *Bazie danych*). W ten sposób ograniczamy niepotrzebny transfer danych między aplikacjami. Możliwe jest także zdalne usuwanie obiektów z aplikacji konkretnych użytkowników, jak i aktualizowanie na życzenie *Serwera* listy posiadanych przez *Aplikacje klienta* obiektów.

Aplikacja *Serwera* – odpowiedzialna za łączenie warstw systemu – sama składa się z „warstw" modelujących komunikację między *Bazą danych* oraz *Konfiguratorem* i *Aplikacją Klienta*.

DatabaseClasses Klasy warstwy *Bazy danych* to przede wszystkim `Database`, po której dziedziczy `DatabaseAdmin` oraz `DatabaseClient`. `DatabaseAdmin` jest odpowiedzialna za udostępnianie głównej części serwera funkcji wywołujących w bazie procedury podtrzymujące informacje o działaniu serwera. Aplikacja *Administratora* nie może wykonywać niebezpiecznych operacji (np. manipulacji przy obiektach) podczas działania serwera. Jednak ze względu na ewentualną możliwość utraty połączenia z serwerem bądź nieprzewidywanego zakończenia się aplikacji *Serwera Baza danych* „sama" po upłynięciu określonego czasu usunie informacje o włączonym *serwerze*. Funkcje klasy `DatabaseAdmin` są podczepione pod `Timer`, który przy starcie wywołuje aplikacja *Serwera*.

DatabaseClient Udostępnia wszystkie funkcje potrzebne do wywoływania operacji bazodanowych zarówno na zlecenie akcji wywołanych przez zewnętrzne aplikacje (*Konfigurator*, *Aplikacja klienta*) jak i wewnętrznych potrzeb *Serwera* (np ustalanie oficjalnej listy plików, czy języków).
Na uwagę zasługuje fakt, iż nigdzie w *Serwerze* nie występuje bezpośrednie zapytanie *Bazy danych* o elementy konkretnych tabel. Poprzez rozbudowanie logiki wbudowanych procedur i widoków w *Bazie danych* każde żądanie dostępu do bazy ogranicza się do wywołania procedury z odpowiednimi parametrami, lub też polecenie wywołania `SELECT *` na zaimplementowanym widoku.

ConnectionClasses Klasy warstwy komunikacyjnej zebrane zostały w folderze `ConnectionClasses`. Za interpretacje komunikatów przychodzących od *Konfiguratora* odpowiada klasa `RegisterConnection`, natomiast *aplikacje Klienta* obsługuje `UserConnection`.

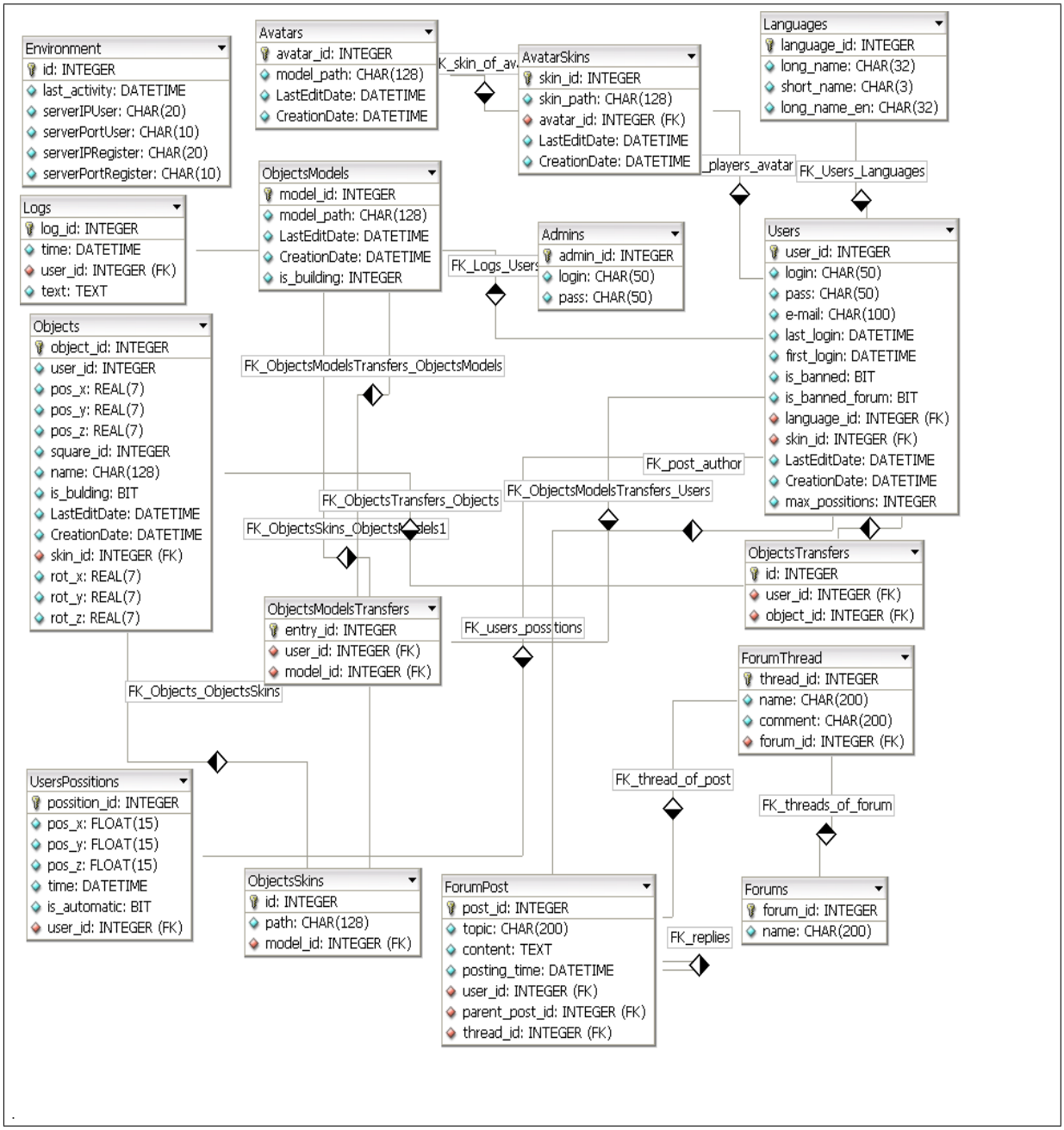
Klasy Łączone W celu unifikacji znaczeń i terminologii używanych w systemie do opisywania obiektów wiele klas wspólnych dla kilku warstw projektu zostało dołączonych jako tzw. „importy". *Serwer* jako warstwa pośrednicząca, z którą wszystkie elementy systemu mają kontakt jest naturalnym miejscem na przechowywanie oryginałów tych klas. I tak pliki zebrane w folderze `CommonClasses` opisują takie obiekty jak język, plik, budynek, obiekt, wektor. Należy przy tym rozróżnić budynek od obiektu: budynek jest identyfikowany na poziomie komunikacji z *Konfiguratorem*, natomiast dla *Aplikacji klienta* jest zwykłym obiektem – nie ma różnicy w wyświetlaniu drzewa i budynku.

Wątki W *Serwerze* tuż po starcie i pomyślnym nawiązaniu połączenia z *Baza danych* wywoływane są 2 osobne wątki. Ich zadaniem jest oczekiwanie na nadchodzące połączenia odpowiednio od *aplikacji Klienckich* i *Konfiguratorów*.

2.5. Baza danych

Podstawowym zadaniem *Bazy danych* jest zapamiętywanie w miarę aktualnego stanu gry. Dodatkowo niektóre z funkcjonalności systemu celowo zostały zaimplementowane w *Bazie danych* (choć mogłyby znajdować się np. w *Serwerze*) ze względu na potrzebę odciążenia *Serwera*. Całość sprowadza się do wystawienia *Serwerowi* API składającego się z 22 Widoków oraz 57 procedur, które *Serwer* prawie „bezmyślnie" wywołuje.
W *Bazie danych* zostały zaszyte następujące funkcjonalności:

- Pamięć o ostatnich pozycjach użytkowników, tak by po wylogowaniu się – „umyślnym" bądź nie – mogli pojawić się w mniej więcej tym samym miejscu na planszy. Odpowiednia procedura w serwerze wywołuje jedynie zakomunikowanie *Bazie danych* nowej pozycji gracza, baza sama decyduje co i kiedy zapamiętać,
- Pamięć o obiektach wysłanych do użytkownika. *Serwer* zadaje bazie pytanie, co ma wysłać danemu użytkownikowi, a baza dostarcza mu gotową listę. Nie wysyłamy informacji o tych samych obiektach wielokrotnie do tych samych użytkowników,
- Pamięć o otwartych sesjach *Serwerów*. Ze względu na potencjalne niebezpieczeństwo ingerencji *Administratora* w strukturę bazy podczas działania serwera Baza sama zarządza dostępem do siebie. *Administrator* wykonując potencjalnie „szkodliwą" operacje pyta się bazy, czy może to zrobić. Zostało to zrealizowane wewnątrz procedur, dostępnych jako interfejs aplikacji *Administratora*.



Rysunek 2.3: Diagram *Bazy danych* – widok pól tabel

Rozdział 3

Protokół i łączenie warstw systemu

Podstawowym zadaniem protokołu i modułów go obsługujących jest połączenie wszystkich kluczowych warstw systemu. Za pomocą 30 typów komunikatów realizowana jest wymiana informacji o:

- obiektach w grze ($Serwer \Leftrightarrow Klient$),
- zmianach stanu awatarów ($Serwer \Leftrightarrow Klient$),
- wiadomościach tekstowych ($Serwer \Leftrightarrow Klient$),
- logowaniu użytkowników (w tym broadcast do pozostałych użytkowników w grze)($Serwer \Leftrightarrow Klient$),
- modelach i skórach używanych w grze ($Serwer \Leftrightarrow Konfigurator$),
- bieżącej wersji serwera ($Serwer \Leftrightarrow Konfigurator$),
- rejestracji użytkowników ($Serwer \Leftrightarrow Konfigurator$),
- rejestracji budynków ($Serwer \Leftrightarrow Konfigurator$).

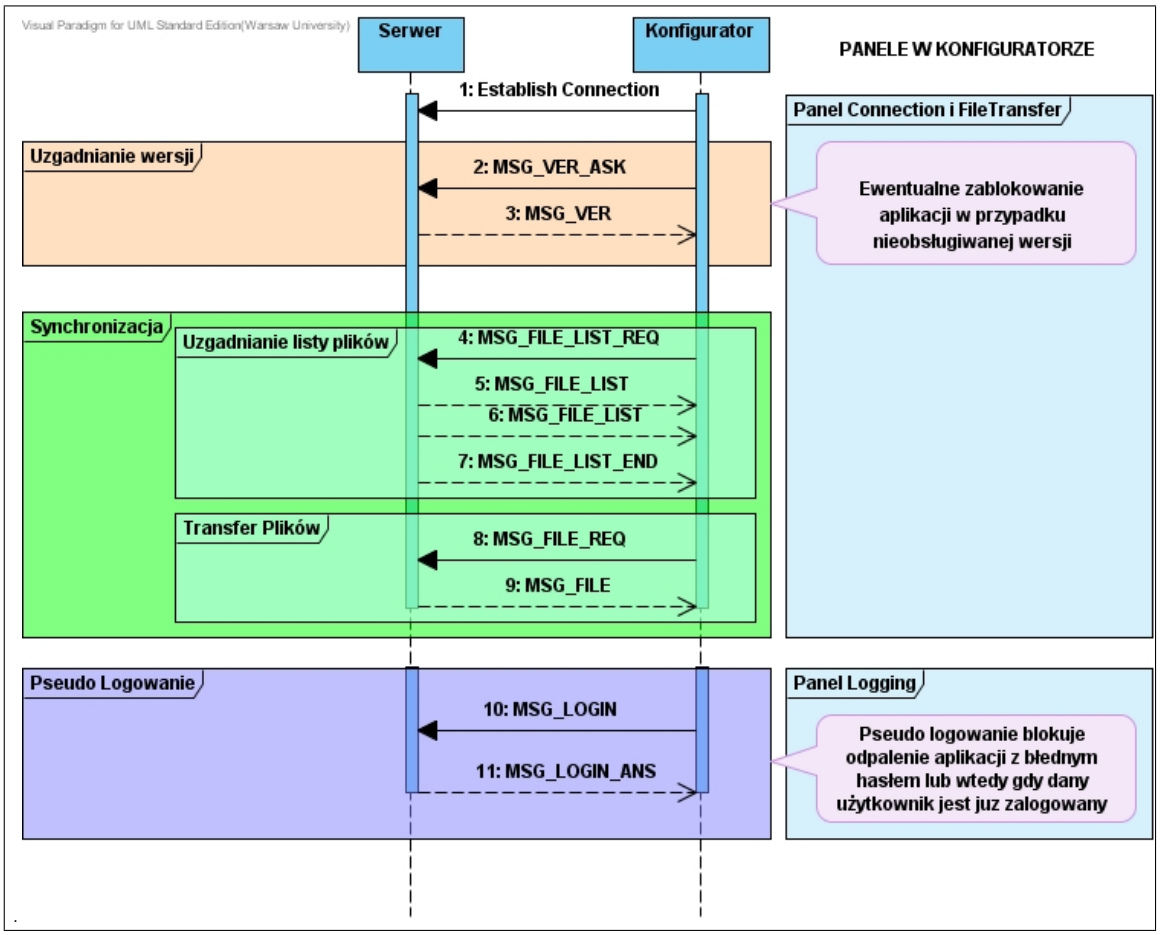
Implementacja. Implementacja protokołu różni się w zależności od warstw.
Aplikacje klienta napisaliśmy w C++ (ze względu na wymogi *engine'u*), a co za tym idzie nie mogliśmy korzystać z wbudowanych mechanizmów takich jak `NetworkStream` (dostępnych w C#). *Konfigurator* z założenia miał obsługiwać prostą komunikacje z *serwerem*, z drugiej zaś strony najbardziej obciążającą ze względu na potencjalnie duży rozmiar transferowanych plików. *Serwer* – napisany w C# – kontroluje przepływ informacji w systemie współbieżnie obsługując połączenia z *Konfiguratorami* i *aplikacjami Klienta*. Kluczowe założenia protokołu spisaliśmy w pliku `Protocole.cs`. Są one zapisywane w projekcie *Serwer*. Pozostałe części systemu, które używają protokołu "linkują" ten plik do własnych zasobów.
W ramach przyjętej praktyki implementacyjnej w poszczególnych aplikacjach, części odpowiedzialne za komunikacje rozdzielono na klasy (przez podział na pliki – `partial class`) odpowiedzialne za wysyłanie (w tym konstruowanie) i odbieranie komunikatów (w tym parsowanie).

Łączenie warstw systemu. Jedną z dwóch form komunikacji, której nie specyfikuje protokół jest przekazanie ustawień środowiskowych z *Konfiguratora* do *aplikacji Klienta*. Znajdują się tam informacje o wybranym przez użytkownika profilu (wybranym do zalogowania), oraz o dostępnych w danej sesji *serwera*, a pobranych przez *Konfigurator* plikach (właściwie ścieżkach do plików) wraz z informacjami o rodzaju danego pliku.

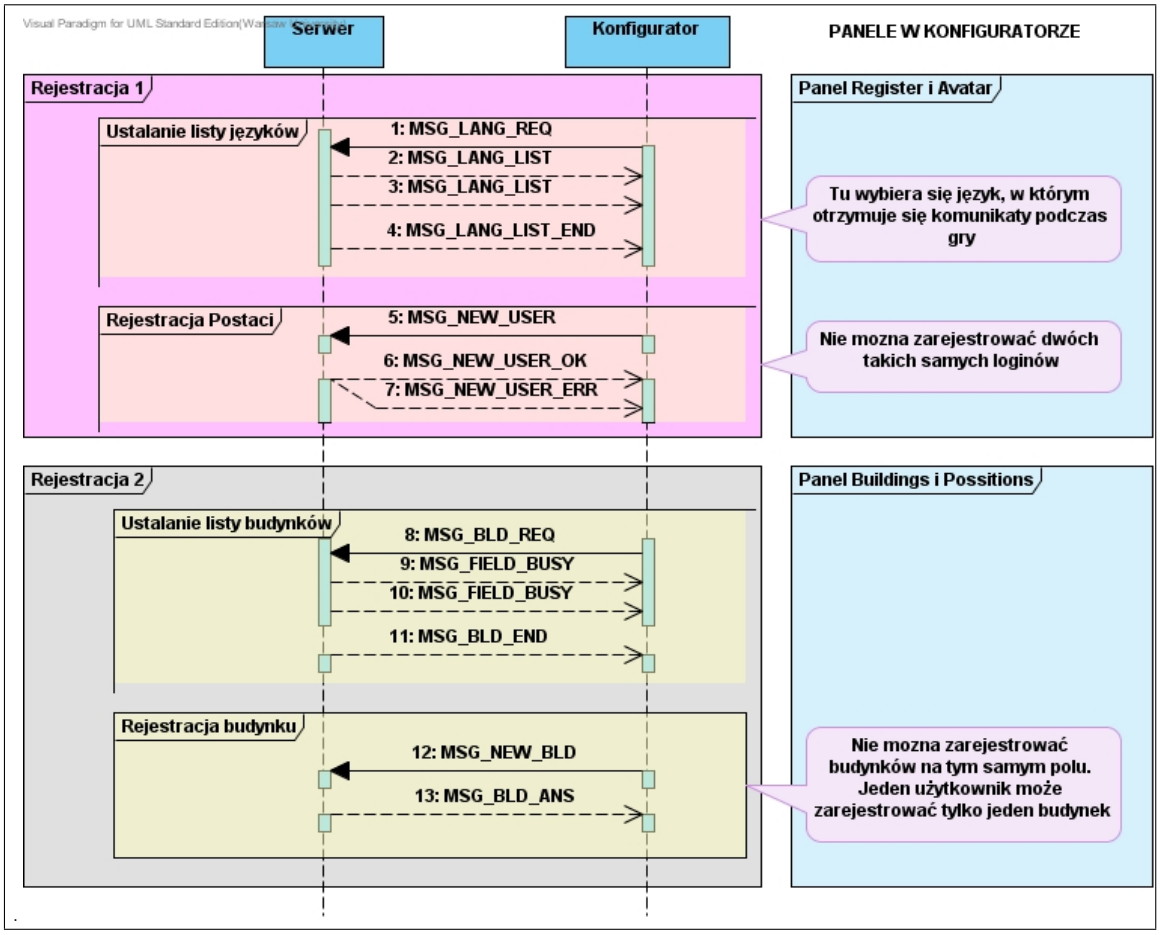
Drugą i ostatnią formą komunikacji niespecyfikowanej przez protokół jest komunikacja *Serwera* z *Baza danych*. Opis przeznaczenia klasy `Database` i klas dziedziczących po niej znajduje się w rozdziale 2.5.

Pozostała komunikacja ($Serwer \Leftrightarrow Konfigurator$) oraz ($Serwer \Leftrightarrow Klient$) realizowana jest przez własny protokół. Ważnym założeniem, które zostało utrzymane jest brak bezpośredniego dostępu do *Bazy danych* aplikacji dostępnych publicznie (*Klient*, *Konfigurator*).

Przykładowe schematy Na rysunku zaprezentowano przykładowe schematy komunikacji i ich przełożenie na logikę aplikacji (w przykładzie jest to aplikacja Konfiguratora).



Rysunek 3.1: Połączenie *Serwer* ⇔ *Konfigurator* – schemat podstawowego użycia (bez rejestracji)



Rysunek 3.2: Połączenie Serwer Konfigurator – schemat rejestracji

Rozdział 4

Podział Prac

Poniżej przedstawiamy szczegółowy podział prac „części implementacyjnej" naszego projektu.

Łukasz Kidziński

Zaprojektowałem i zaimplementowałem Klienta. Współprojektowałem ogólną koncepcję protokołu. Dobrałem technologie odpowiednie do naszego projektu. Dbałem o przejrzysty interfejs WWW, Konfiguratora i Klienta.
Przygotowałem środowisko Visual Studio do współpracy z silnikiem graficznym z którego korzystamy.

1. **Klient**
   Aplikacja klienta została zaprojektowana i napisana w całości przeze mnie.
2. **Konfigurator**
   - Wdrożenie silnika MOGRE
   - Klasa odpowiadająca za proste pakowanie/rozpakowywanie plików
   - Interfejs użytkownika
   - Koncepcja podziału na klasy
   - Wdrożenie kilku wersji językowych konfiguratora
3. **Serwer**
   Przygotowanie środowiska i współprojektowanie ogólnej koncepcji komunikacji z klientem. Bardzo mały udział w implementacji – jedynie na samym początku.
4. **Baza danych**
   Współprojektowanie pierwszej wersji. Nie brałem udziału w implementacji.
5. **Strona WWW**
   Przygotowanie grafiki i kodu html.
6. **Inne**
   - Organizowanie grafiki, serwera i wyszukiwanie sponsorów i partnerów.
   - Testy klienta, konfiguratora i serwera.

Paweł Bedyński

Zajmowałem się głównie komunikacją pomiędzy warstwami. Zaimplementowałem prawie cały Serwer oraz znaczne części Bazy danych i Konfiguratora.

1. **Baza Danych**
   Zaimplementowałem całą bazę danych która zdaliśmy 30.03.2009. Po tym terminie najwięcej zmieniło się ze względu na komunikacje Baza – Admin (dużo nowych procedur), jednak zasadniczy schemat bazy nie uległ dużej zmianie.
   - założenie wszystkich tabel
   - triggery na tabele w bazie
   - procedury odpowiadających za komunikacje Baza – Serwer
   - większość widoków
   - wstępne wersje procedur odpowiadających za kontrolowanie operacji współbieżnych pomiędzy Serwerem a Administratorem
2. **Serwer**
   Z aplikacji Serwera napisałem wszystko poza: wstępną wersją podziału na wątki oraz wstępną wersją obsługi nawiązywania i zamykania połączeń.
3. **Konfigurator**
   Z aplikacji Konfiguratora napisałem:
   - pełną obsługę połączenia z serwerem (w tym implementacja logiki odbieranych i wysyłanych komunikatów)
   - moduł odpowiedzialny za zarządzanie profilami użytkowników
   - moduł odpowiedzialny za transfer plików i synchronizację z serwerem (ostateczny wygląd oraz podczepienie synchronizacji pod klasę *BackgroundWorker* napisał Łukasz)
   - obsługę SerwerLoga (gdzie wyświetlają się informacje o postępie operacji wykonywanych z wykorzystaniem komunikacji z Serwerem)
   - cześć prac nad wyglądem konfiguratora.
4. **Instalator**
   Wstępna wersja instalatora stworzonego przy użyciu programu **inno Setup**

Artur Matan

Zajmowałem się głównie Aplikacją Administratora. Poza tym miałem swój udział w Bazie Danych, stronie WWW oraz w niewielkim stopniu w Serwerze.

1. **Aplikacja Administratora**
   Aplikacja Administratora została zaprojektowana i napisana w całości przeze mnie (wyjątek – „packowanie plików" – wspólne z Konfiguratorem)
2. **Baza Danych**
   W dużej mierze zaprojektowałem i całkowicie zaimplementowałem część Bazy Danych odpowiedzialną za komunikacje Baza – Admin:
   - procedury, pozwalające na proste i przejrzyste działania Administratora
   - widoki
   - nowe wersje procedur odpowiadających za kontrolowanie operacji współbieżnych pomiędzy Serwerem a Administratorem (synchronizacja)
3. **Strona WWW**
   Mój udział – zaimplementowałem tłumaczenia.
4. **Serwer**
   Minimalny udział w implementacji – prototyp serwera.
5. **Inne**

   - Drobne poprawki w innych aplikacjach.
   - Niewielki udział w testowaniu Klienta/Konfiguratora.

Rozdział 5

Opis zawartości załączonej płyty DVD

Na dołączonej do niniejszej pracy płycie DVD znajduje się:

- *Instalator* (wersja dostępna publicznie poprzez stronę WWW) w folderze:

  `/klient/setup`

- Kod *Aplikacji klienta* w folderze:

  `/klient/src/klient`

- Kod *Konfiguratora* w folderze:

  `/klient/src/konfigurator`

- Kod *Serwera* w folderze:

  `/serwer/src/serwer`

- Kod Aplikacji *Administratora* w folderze:

  `/serwer/src/admin`

- Skrypty tworzące *Bazę danych* MSSQL (tabele, procedury, triggery, widoki) w folderze:

  `/serwer/src/baza`

- Instalator *Serwera* (wraz z *Aplikacją administratora*)

  `/serwer/setup`

- Kod strony WWW w folderze:

  `/serwer/src/www`

- Dokumentacje projektu w folderze:

  `/doc`

Dodatek A

Szczegółowy opis wymienianych komunikatów

- Komunikaty Pseudo Logowania ($Serwer \Leftrightarrow Konfigurator$)
  public const int MSG_LOGIN = 0x000100; – logowanie użytkownika
  public const int MSG_LOGIN_ANS = 0x000103; – odpowiedź serwera
- Komunikaty do ustalenia wersji ($Serwer \Leftrightarrow Konfigurator$)
  public const int MSG_VER_ASK = 0x000F00; – pytanie o wersje serwera
  public const int MSG_VER = 0x000F10; – odpowiedz serwera
- Komunikaty Logowania ($Serwer \Leftrightarrow Klient$)
  public const int MSG_LOGIN = 0x000100; – logowanie użytkownika
  public const int MSG_LOGIN_OK = 0x000101; – komunikat o poprawnym logowaniu.
  public const int MSG_LOGIN_ERROR = 0x000102; – komunikat o błędzie w logowaniu
- Komunikaty rozgłaszania logowania ($Serwer \Leftrightarrow Klient$)
  public const int MSG_USER_LOGGED_IN = 0x000120; – informacja dla innych użytkowników o nowo zalogowanym użytkowniku.
- Komunikaty tworzenia nowego użytkownika ($Serwer \Leftrightarrow Konfigurator$)
  public const int MSG_NEW_USER = 0x000110; – próba zarejestrowania nowego użytkownika.
  public const int MSG_NEW_USER_OK = 0x000111; – komunikat do konfiguratora informujący o poprawnym zarejestrowaniu użytkownika.
  public const int MSG_NEW_USER_ERR = 0x000112; – błąd przy dodaniu nowego użytkownika.
- Komunikaty dotyczące zmiany stanu awatarów (np. chodzenie) ($Serwer \Leftrightarrow Klient$)
  public const int MSG_NEW_STATE = 0x000200; – informacja do serwera "zmieniam swój stan"
  public const int MSG_USER_NEW_STATE = 0x000300; – broadcast do pozostałych użytkowników że ktoś zmienił stan.
- Komunikaty dotyczące rozmów ($Serwer \Leftrightarrow Klient$)
  public const int MSG_CHAT = 0x000400; – komunikat rozmowy przychodzący do serwera.
  public const int MSG_USER_CHAT = 0x000500; – broadcast do wybranych użytkowników łącznie z tłumaczeniem.
- Komunikaty dotyczące budynków ($Serwer \Leftrightarrow Konfigurator$)
  public const int MSG_NEW_BLD = 0x000700; – próba stworzenia nowego budynku.
  public const int MSG_BLD_ANS = 0x000740; – odpowiedź serwera na próbę stworzenia nowego budynku.
  public const int MSG_BLD_REQ = 0x000760; – prośba o przesłanie listy budynków.
  public const int MSG_BLD_END = 0x000770; – koniec przesyłania listy budynków.
  public const int MSG_BLD_NEW_FIELD = 0x000780; – informacja o zajętym polu (budynku) wraz z informacjami o danym budynku.
- Komunikaty dotyczące pozycji awatarów na mapie ($Serwer \Leftrightarrow Klient$)
  public const int MSG_USER_POSS_REQ = 0x000B00; – żądanie przesłania pozycji dla awatara
  public const int MSG_USER_POSS_ANS = 0x000B10; – pozycja awatara
- Komunikaty odpowiedzialne za transfer plików ($Serwer \Leftrightarrow Konfigurator$)

public const int MSG_FILE_LIST = 0x000C00; – informacje („nagłówki”) o pojedynczym pliku.
public const int MSG_FILE_LIST_END = 0x000C10; – zakończenie przesyłania informacji o plikach.
public const int MSG_FILE_REQ = 0x000C20; żądanie przesłania listy plików.
public const int MSG_FILE = 0x000C30; – żądanie przesłania pojedynczego pliku

- Przesyłanie informacji o obiektach w grze ($Serwer \Leftrightarrow Klient$)

  public const int MSG_NEW_OBJECT = 0x000D00; – informacja o nowym obiekcie w grze
  public const int MSG_DEL_OBJECT = 0x000D10; – żądanie usunięcia obiektu z gry

- Przesyłanie informacji o dostępnych językach ($Serwer \Leftrightarrow Konfigurator$)
  public const int MSG_LANG_REQ = 0x000E00; – pytanie o liste języków
  public const int MSG_LANG_LIST = 0x000E10; – pojedynczy dostępny język
  public const int MSG_LANG_LIST_END = 0x000E20; – koniec wysyłania listy języków

Dodatek B

Podsumowanie

- **Wielkość systemu mierzona w linijkach kodu poszczególnych aplikacji** (stan z 24.05.2009)

Aplikacja	**Linijki kodu**
Klient	3891
Konfigurator	2613
Serwer	3592
Administrator	1767
Baza danych	7309
WWW	406
Instalator	67
razem	**19645**

- **Dostępność aplikacji**
  Instalator jest dostępny pod adresem **http://hepiplanet.com**. Gwarantujemy dostępność strony do 1.05.2010, a dostępność *Serwera* do 1.07.2009.

- **Zgłoszenia w konkursach**
  Prezentowany projekt został, lub zostanie zgłoszony w następujących konkursach:

  1. UBS AG London (konkurs wewnątrzwydziałowy)
  2. Microsoft Imagine Cup
  3. ...

Bibliografia

[1] Torus Knot Software Ltd, *OGRE API Reference*, http://www.ogre3d.org/docs/api/html/.

[2] Gregory Junker, *Pro OGRE 3D Programming*, APRESS 2006.

[3] Microsoft, *Microsoft Developers Network*, http://msdn.microsoft.com/pl-pl/default.aspx.