Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie

Wydział Informatyki, Elektroniki i Telekomunikacji

Kierunek: Informatyka

DOKUMENTACJA PROJEKTU

Budowa zintegrowanego środowiska dostępowego do bazy danych w celu usprawnienia przetwarzania danych w eksperymentach fizyki wysokich energii

Autorzy: Bartłomiej Alberski, Michał Idzik, Bartosz Niemczura

Opiekun pracy: dr inż. Łukasz Czekierda

Prowadzący: dr inż. Tomasz Szydło

Opiekun techniczny: mgr inż. Przemysław Wyszkowski

Kraków, styczeń 2013

1. Cel projektu

TOTEM Unified Database Access Service (TUDAS) to system, którego zadaniem jest pomoc w efektywnym zarządzaniu dostępem do danych z eksperymentu TOTEM [1] w ośrodku badawczym CERN[2]. Dodatkowo, ma on zapewnić łatwą rozszerzalność pod kątem nowych usług oraz nieskomplikowany sposób obsługi przez użytkownika końcowego.

Aby osiągnąć taki rezultat, zostało stworzone złożone, w wielu aspektach transparentne dla użytkownika rozwiązanie, zapewniające niezawodną komunikację z TOTEM Offline Database[3].

Celem projektu jest dostarczenie rozwiązania umożliwiającego wygodną pracę z bazą danych, bez konieczności pisania przez użytkownika komend SQL, a nawet bez wiedzy o strukturze bazy. Jednym z głównych priorytetów jest jak największe uproszczenie klienckiej strony systemu. W rezultacie stworzony został w pełni funkcjonalny, wysokopoziomowy interfejs programistyczny. Wspomniane API jest zunifikowane i bazuje na podstawowych regułach programowania zorientowanego obiektowo. Każdy z użytkowników ma możliwość korzystania z systemu wykorzystując preferowany przez siebie język programowania - system umożliwia tworzenie aplikacji klienckich w Javie, Pythonie oraz C++.

Zapewnienie efektywnego dostępu do bazy danych nie jest ostatnim wymaganiem stawianym przed systemem. TUDAS ma także dostarczyć mechanizmy bezpieczeństwa gwarantujące, że dane będą chronione przed nieautoryzowanymi zapytaniami. W tym celu, projektowany system został zintegrowany z CERN Single Sign On[4], serwisem globalnie wykorzystywanym w całym ośrodku, który gwarantuje spełnienie wszelkich norm bezpieczeństwa. Dodatkowo, każda bazodanowa operacja jest logowana, aby administratorzy systemu mogli nadzorować działania użytkowników.

Motywacją do realizacji systemu było dostarczenie produktu, który ułatwi pracę naukowcom oraz przyczyni się do ich wydajniejszej pracy. Pobieranie danych z bazy TOTEM'u przy wykorzystaniu tworzonego narzędzia znacząco ułatwi sposób wymiany informacji oraz dostarczy środków dla nadania odpowiednich relacji między danymi eksperymentalnymi.

---

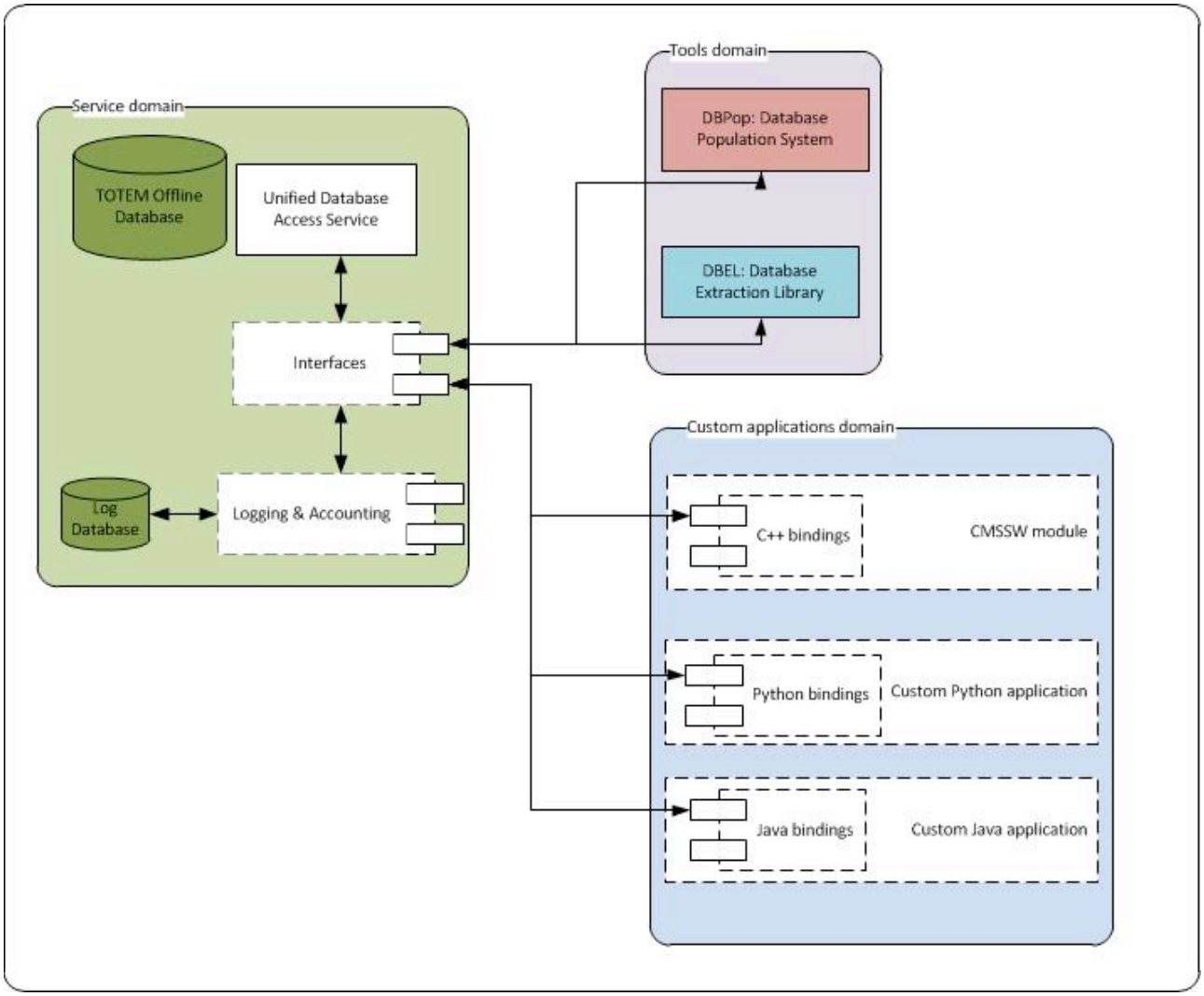
1 TOTEM (ang. TOTal Elastic and diffractive cross section Measurement) - eksperyment działajacy "przy LHC(5)", http://totem.web.cern.ch.

2 CERN (The European Organization for Nuclear Research) - ośrodek naukowo-badawczy położony na północno-zachodnich przedmieściach Genewy na granicy Szwajcarii i Francji, http://cern.ch

3 TOTEM Offline Database - baza danych eksperymentu TOTEM wykorzystywana do przechowywania wyników pomiarów.

4 CERN Single Sign On - system wykorzystywany w CERN'ie umożliwiający jednorazowe zalogowanie się do usługi sieciowej i uzyskania dostępu do wszystkich autoryzowanych zasobów zgodnych z tą usługą

5. LHC (ang. Large Hadron Collider) - największy na swiecie akcelerator czastek elementarnych, znajdujący się w CERN'ie w pobliżu Genewy, http://lhc.web.cern.ch.



Rysunek 1: Wizja sytemu.

2. Wykorzystywane technologie

Podczas fazy projektowania rozwiązania dokonany został wybór technologii, które najlepiej spełniały warunki stawiane wobec systemu. Badane technologie dotyczyły modułu bazodanowego (mające na celu m.in. poprawę wydajności oraz bezpieczeństwa operacji bazodanowych), modułu serwera i klientów (m.in. technologia middleware, serwer aplikacji). Szczegółowe zestawienie wybranych technologii i narzędzi zostało przedstawione poniżej.

2.1. Strona bazodanowa

W trakcie prac nad mechanizmem umożliwiającym przeprowadzanie operacji na bazie danych użyto technologie, które znacząco ułatwiły pracę nad projektem oraz przyczyniły się do hierarchizacji kodu. Wśród nich należy zwrócić uwagę na:

- **Spring IoC** - umożliwił deklaratywne wsztrzykiwanie implementacji interfejsów bazodanowych i odseparowanie kodu realizującego procedury biznesowe od logiki konfiguracyjnej,
- **Spring AOP** - ułatwił zaimplementowanie funkcjonalności pozwalającej na dynamiczny dobór konta bazodanowego w zależności od realizowanych operacji bazodanowych przez usługę,
- **Ehcache** - wiele z danych zawartych w bazie danych nie zmienia się zbyt często np. prawa użytkowników czy schemat budowy systemu detektorów, dlatego warto te dane przechowywać w pamięci podręcznej. Do realizacji wspomnianej optymalizacji została użyta biblioteka Ehcache.
- **AspectJ** - narzędzie umożliwiło deklaratywną realizację transakcyjności operacji,
- **Commons-dbcp** - narzędzie umożliwiło przechowywanie aktywnego połączenia w oczekiwaniu na kolejne zapytanie bazodanowe.

2.2. Strona serwerowa

Serwer, umiejscowiony pomiędzy zapytaniami klientów a ich realizacją w części bazodanowej stanowi miejsce, w kluczowy sposób wpływające na efektywność pracy całego systemu. Dlatego też, aby zniwelować efekt wąskiego gardła, konieczne było znalezienie odpowiedniej technologii, spełniającej szereg wymagań. Po przeprowadzeniu wnikliwych badań (opisanych w Dokumentacji procesowej), zostały wybrane następujące technologie:

- **ZeroC ICE** - middleware, umożliwiający zaawansowaną obslugę zdalnych wywołań metod, bezpieczne przetwarzanie wielu zapytań klientów jednocześnie, mechanizmy równoważnia obciążenia i zarządzania obiektami [5] (Servant Locator[6]),
- **Spring IoC** - deklaratywne wstrzykiwanie implementacji zdalnych interfejsów ICE i mechanizmów bezpieczeństwa, bez konieczności programistycznego budowania powiązań między obiektami systemu,
- **AspectJ** - wygodne, transparentne dla programisty, realizowane aspektowo mechanizmy logowania informacji o wywołaniach klienta, a także warstwa autoryzacji zapytań.

2.3. Strona kliencka

W przypadku strony klienckiej, należało znaleźć technologię która umożliwiłaby komunikację z serwerem bez względu na język programowania, w jakim została stworzona biblioteka kliencka. Dokładny opis wyboru technologii middleware został opisany w "Dokumentacji procesowej". Stawiane przed tą częścią systemu warunki spełnił middleware ICE, natomiast narzędzia go wspomagające pozwoliły na wygenerowanie kodu dla proxy klientów i serwera.
Dodatkowo, w przypadku klienta napisanego w języku Java, technologia AspectJ pozwoliła na uzupełnienie mechanizmu autoryzacji/uwierzytelniania. W przypadku klientów C++ i Python podobna funkcjonalność została zrealizowana przy użyciu specyficznych konstrukcji dla każdego z wspomnianych języków programowania.

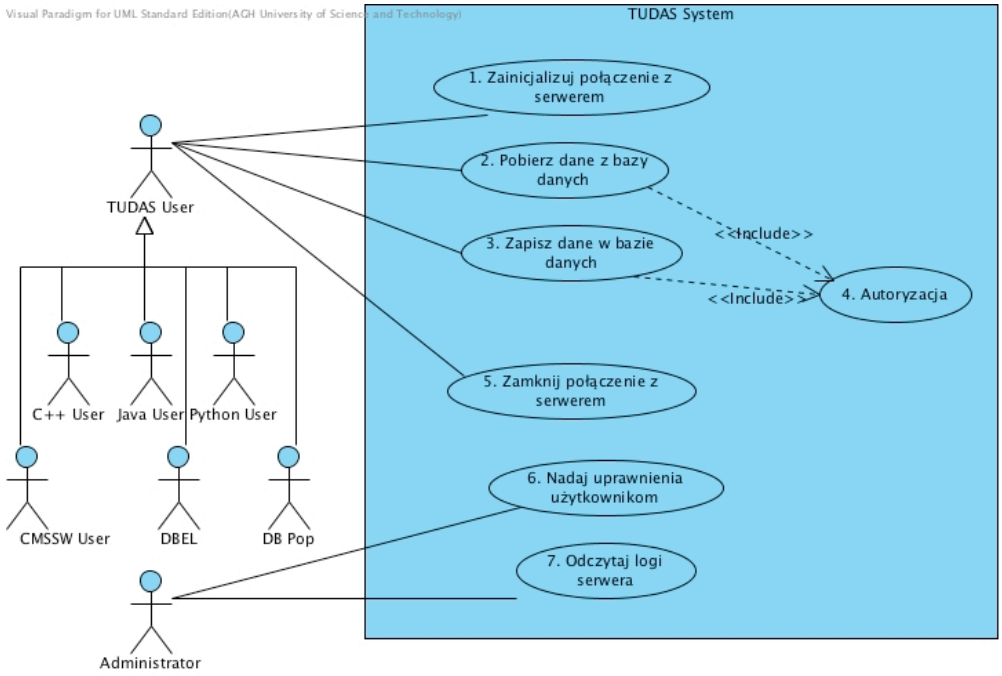
5 Serwant - klasa, której instancja dostarcza realizacji pewnej ściśle okreslonej grupy usług.

6 Servant Locator - obiekt odpowiedzialny za utworzenie odpowiedniego serwanta i przekazanie do niego wywołania w przypadku gdy nie istnieje serwant realizujący żądanie.

3. Wymagania

Większość wymagań funkcjonalnych odnosi się do metod odczytu i zapisu danych w bazie eksperymentu TOTEM. Wymagania systemu zostały ustalone podczas wielu wywiadów z naukowcami pracującymi w CERN'ie. Równocześnie z wymaganiami funkcjonalnymi zostały zebrane uwagi na temat jakości działania systemu - wymagania niefunkcjonalne. Użytkownikami systemu są naukowcy tworzący aplikacje w jednym z trzech języków programowania (C++, Java, Python), istniejące systemy korzystające z bazy danych: DBPop[7], DBEL[8], jak również moduły osadzone we frameworku CMSSW [9] używanego w CERN do obliczeń fizycznych. Zidentyfikowane przypadki użycia systemu TUDAS oraz wymagania funkcjonalne i niefunkcjonalne zostały szczegółowo omówione poniżej.

3.1. Przypadki użycia



Rysunek 2 : „Diagram przypadków użycia systemu TUDAS".

7 DBPop (ang. Totem Database POPulation System) - system służący do populacji(zapisu) danych do bazy eksperymentu TOTEM, https://twiki.cern.ch/twiki/bin/view/TOTEM/CompDBSoftwareDBPOP

8 DBEL(ang. Database Extraction Library) - system służacy do eksportowania danych z bazy ekesperymentu TOTEM do plików w formacie XML , https://twiki.cern.ch/twiki/bin/view/TOTEM/CompDBDbelTotemDataBaseExtractionLibrary

9 CMSSW (ang. CMS(10) Software) - framework używany w eksperymentach związanych z detektorem CMS, https://twiki.cern.ch/twiki/bin/view/CMS/WorkBook

10 CMS (ang. Compact Muon Solenoid) - detektor cząstek elementarnych LHC, tą nazwą określany jest również kolejny z eksperymentów przeprowadzanych w LHC w CERN'ie, http://cms.web.cern.ch/.

3.1.1. Inicjalizacja połączenia z serwerem

Użytkownik systemu wywołuje metodę inicjalizującą połączenie z serwerem TUDAS. Sprawdzana jest poprawność konfiguracji Workspace TUDAS, przygotowany jest ticket[10] na podstawie nazwy użytkownika i hasła. Po pomyślej inicjalizacji, użytkownik może korzystać z serwisów oferowanych przez system TUDAS.

3.1.2. Pobierz dane z bazy danych

Dane, o które zostało zgłoszone żądanie, są pobierane z bazy danym TOTEM Offline Database i udostępniane użytkownikowi. Użytkownik jest autoryzowany w celu sprawdzenia, czy może on otrzymać dane, o które zgłosił żądanie.

3.1. 3. Zapisz dane w bazie danych

Użytkownik zapisuje dane eksperymentu w bazie danych TOTEM Offline Database. Użytkownik jest autoryzowany w systemie w celu sprawdzenia, czy ma uprawnienia pozwalające na zapis danych do bazy.

3.1. 4. Autoryzacja

Sprawdzana jest poprawność ticketu, który użytkownik posiada we własnym workspace'ie.

3.1. 5. Zamknij połączenie z serwerem

Użytkownik zamyka połączenie z serwerem systemu TUDAS.

3.1. 6. Nadaj uprawnienia użytkownikom

Administrator nadaje uprawnienia użytkownikom, aby wskazać którzy z nich mogą zapisywać/ odczytywać konkretne dane z bazy danych.

3.1. 7. Odczytaj logi serwera

Administrator może odczytać logi serwera w celu zidentyfikowania awarii lub sprawdzenia aktywności użytkowników.

10 ang. Ticket - zaświadczenie uprawniające do korzystania z określonych usług systemu.

3.2. Wymagania funkcjonalne

- system powinien pozwalać na zainicjalizowanie i zakończenie połączenia z serwerem TUDAS,
- możliwość zapisu i odczytu danych eksperymentu:
  - dane dotyczące pozycji aparatury pomiarowej Roman Pot[11],
  - dane dotyczące konkretnego powtórzenia eksperymentu (Run[12] Information),
  - dane dotyczące optyki używanej podczas eksperymentu,
  - dane dotyczące świetlności podczas eksperymentu w kolejnych punktach pomiaru,
  - dane dotyczące ogólnych warunków panujących w tunelu podczas przeprowadzania eksperymentu (temperatura, ciśnienie);
- możliwość nadawania i usuwania uprawnień użytkowników do operacji bazodanowych,
- możliwość odczytywania, którzy użytkownicy mają dostęp do danych oraz ilości czasu jaka potrzebna była na zrealizowanie żądania.

3.3. Wymagania niefunkcjonalne

- obsługa stosunkowo niewielkiej ilości użytkowników (kilku, kilkunastu) zarządzającymi bardzo dużą ilością danych (kilkaset MB/s),
- integracja z systemem CERN - Single Sign On w celu pobrania informacji o użytkowniku,
- bezpieczeństwo systemu - uwierzytelnianie i autoryzacja,
- integracja z bazą danych Oracle,
- możliwość korzystania z systemu wewnątrz frameworku CMSSW,
- warstwa bazy danych powinna realizować mechanizm "Connection Pooling[13]",
- system powinien być dobrze udokumentowany: zawierać dokumentację użytkownika, dewelopera oraz techniczny opis architektury systemu,
- przygotowanie na przyszły rozwój - łatwe dodawanie nowych serwisów, manager'ów [14] danych,
- umożliwić łatwą integrację z istniejącymi systemami bazy danych np.: DBPop, DBEL,
- prostota sygnatur metod udostępnianych w ramach API bibliotek klienckich,
- wydajny transfer danych,
- możliwość przyszłej klasteryzacji serwera.

---

11 ang. Roman Pot - nazwa ruchomego typu detektora umożliwiającego pomiar przekroju czynnego dwóch cząstek w akceleratorze cząstek elementarnych.

12 ang. Run - krótki okres czasu, w którym pobierana są dane z eksperymentu.

13 ang. Connection Pooling - mechanizm przechowywania w pamięci podręcznej aktywnego połączenia do bazy danych.

14 ang. Manager - serwis, usługa skupiająca metody.

4. Koncepcja rozwiązania

Produkt jest podzielony na trzy wyraźnie odseparowane części:

- część bazodanową,
- część serwerową,
- część kliencką.

Każdy z nich, uniezależniony od pozostałych części systemu, ma wyraźnie określoną odpowiedzialność.

4.1. Moduły klienckie

Moduły klienckie zostały zrealizowane w trzech językach programowania: C++, Java, Python. Użytkowanie każdego z nich jest identyczne, ponieważ istnieje zgodność na poziomie interfejsów wystawianych dla aplikacji klienckich. Każdy z tych modułów dostarczony jest w postaci biblioteki specyficznej dla danego języka programowania.

Do zadań części klienckich należy dostarczenie wysokopoziomowego interfejsu do realizacji określonych usług.

4.2. Moduł serwerowy

Moduł serwerowy odpowiada za odbieranie żądań od aplikacji klienckich, ich interpretacje, wstępne przetworzenie oraz przekazanie wywołania do części bazodanowej.

Serwer dostarczony jest w postaci pliku JAR, z wszystkimi zależnościami.

4.3. Moduł bazodanowy

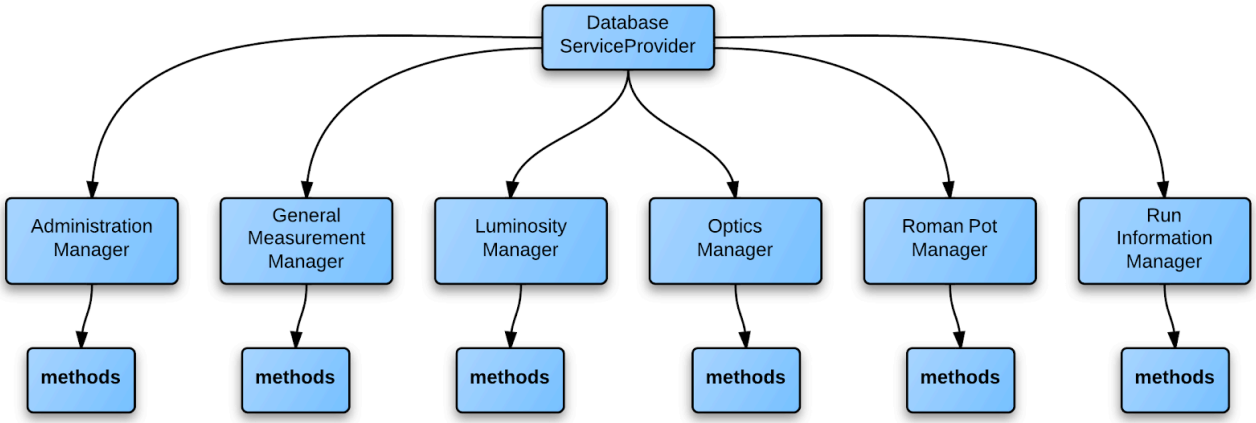
Moduł składa się z dwóch części, z których każda spełnia ściśle określone zadania. Pierwsza z nich zawiera interfejsy specyfikujące komunikację pomiędzy częścią serwerową a modułem dostępowym (databaseInterfaces), natomiast druga z nich (databaseAccess) odpowiada za dostarczenie usług pozwalających na przeprowadzanie operacji bazodanowych takich jak: wyszukiwanie, modyfikowanie oraz zapisywanie danych.

4.4. Udostępnianie usług

Jednym z wymagań stawianych wobec systemu jest prostota obsługi. W tym celu należało dostarczyć proste rozwiązanie, zapewniające ustrukturyzowaną hierarchię oraz otwarte na rozwój. W celu zapewnienia wspomnianych wymagań sposób dostarczania usług do klientów został zrealizowany z wykorzystaniem wzorca service-provider[15].

15. ang. service - provider - wzorzec projektowy pozwalający na scentralizowanie i ustrukturalizowanie dostępu do usług.

Każda z usług zlokalizowana jest w hierarchicznie zbudowanej strukturze:



Rysunek 3: „Hierarchiczna struktura usług oferowanych przez system".

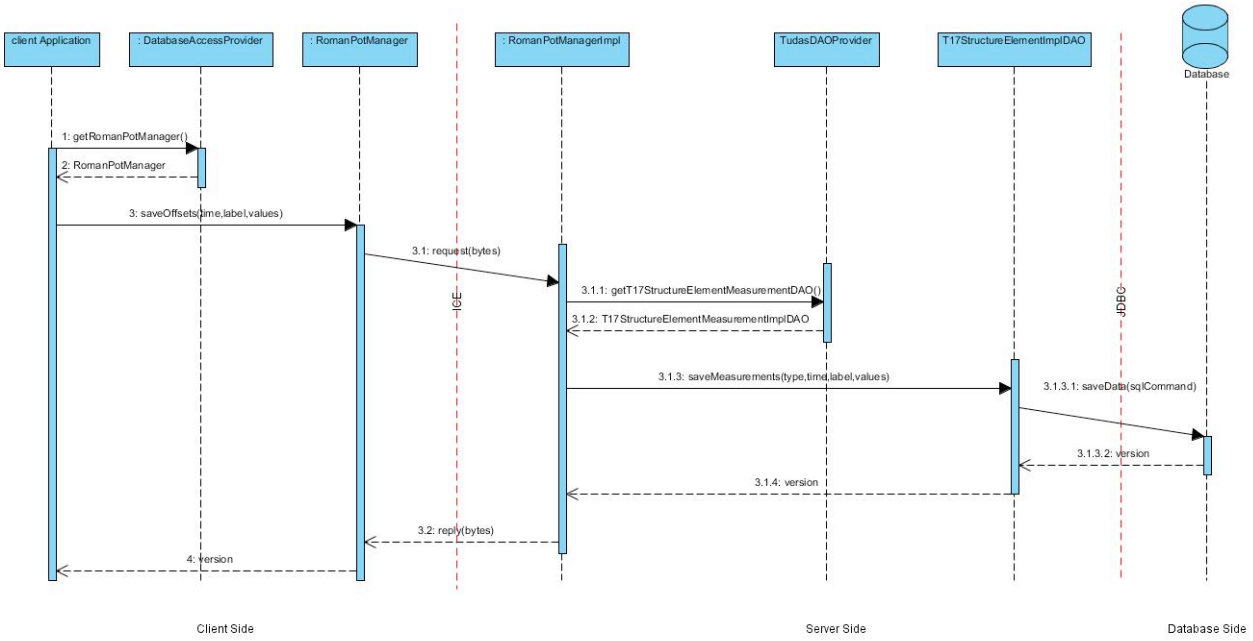
Na szczycie hierarchii znajduje się Database Service Provider, który udostępnia usługi zebrane w logiczne grupy. Poniżej znajdują się zarządcy usług, których zadaniem jest dostarczenie ściśle określonej funkcjonalności. Na samym dole struktury można odnaleźć metody realizujące wymagane zadania.

4.5. Opis usług

Rodzaj oferowanych usług został zdeterminowany przez wymagania postawione przed systemem. Z tego powodu usługi zostały podzielone w logiczne grupy odzwierciedlające rzeczywisty podział funkcjonalności.
Wyodrębnione grupy usług skupiane wokół zarządców/dostarczycieli usług:

- AdministrationManager - grupa usług skupiająca metody związane z zarządzaniem użytkownikami systemu. Pozwala na dodanie, modyfikacje oraz usunięcie uprawnień każdemu z użytkowników systemu,
- General Measurement Manager - grupa metod realizująca funkcjonalności związane z operowaniem na ogólnych danych(ang. General Measurements),
- Luminosity Manager - zbiór usług realizujących zadania związane z zapisem, wyszukiwaniem danych dotyczących świetlności (Luminosity),
- Optics Manager - zestaw funkcjonalności odpowiedzialnych za zapis, wyszukiwanie, oraz listowanie danych związanych z optyką (optics),
- Roman Pot Manager - zawiera kilka metod umożliwiających zapis oraz odczyt danych związanych z detektorami typu Roman Pot,
- Run Information Manager - grupa usług dostarczających możliwości utrwalenia danych związanych z informacją o Run'ie.

4.6. Przykład wywołania



Rysunek 3: „Przykładowe wywołanie. Diagram nie prezentuje czynności związnaych z procesem autoryzacji i uwierzytelniania."

1,2 - Aplikacja kliencka zgłasza żądanie otrzymania RomanPotManager'a do Database Access Providera (DAP). DAP zwraca nowego(lub wcześniej zapamiętanego) manager'a o danym typie.

3.1 - Zwrócony manager marshalluje[16] dane do strumienia bajtów z wykorzystaniem mechanizmu ICE.

3.1.1 - 3.1.2 - Obiekt serwerowy zwraca się z żądaniem do DAOProvider'a o odpowiedni obiekt DAO[17]. DAOProvider zwraca wcześniej zapamiętany obiekt danego typu (lub nowy jeżeli nie było wcześniej żadnego żądania o ten rodzaj obiektu).

3.1.3 - Implementacja obiektu manager'a wywołuje odpowiednią metodę na wcześniej zwróconym obiekcie DAO.

3.1.3.1 - Odpowiednie zapytanie jest wysyłane do bazy danych z wykorzystaniem mechanizm JDBC

3.1.3.2 - Baza danych po wykonaniu zapytania zwraca jego rezultat do obiektu DAO.

3.1.4 - Obiekt DAO interpretuje rezultat wywołania i zwraca go do manager'a po stronie serwerowej.

3.2 - Manager marshalluje odpowiedź od bazy danych i wysyła ją do managera po stronie aplikacji klienckiej

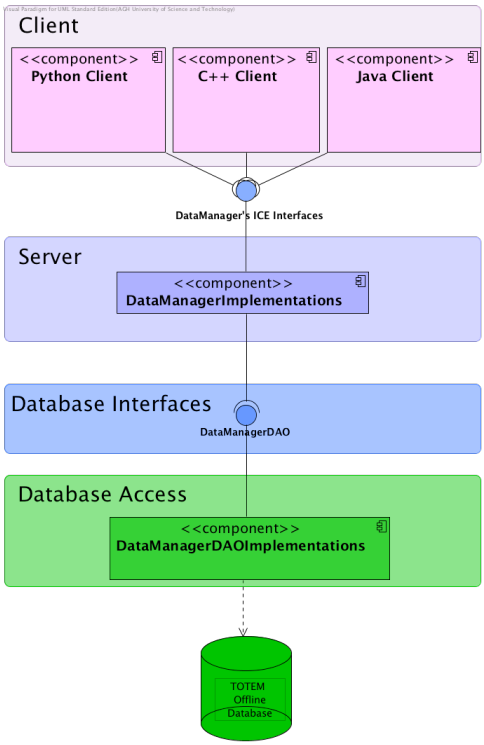
4 - Aplikacja kliencka otrzymuje numer wersji pod którą zostały zapisane dane.

16. Marhalling - proces zamiany obiekty w inny prostszy format(strumień bajtów) w celu przesłania go przez sieć

17. DAO (database access object) - wzorzec projektowy pozwalający na odseparownie logiki persystencji od logiki czysto biznesowej

5. Architektura systemu

System został zbudowany w oparciu o architekturę trójwarstwową, w której interfejs użytkownika, przetwarzanie danych oraz ich składowanie rozwijane są w oddzielnych modułach. Pierwszą warstwą jest moduł klienta, w którym udostępniane są interfejsy do zarządzania danymi. Drugą warstwę stanowi moduł serwera, który komunikuje się z klientem za pośrednictwem mechanizmu proxy ICE'a. Trzecią wartstwę stanowi moduł bazodanowy, który jest odseparowany od modułu serwera dobrze zdefiniowanym interfejsem. Moduł bazodanowy operuje na bazie danych wykorzystując sterownik JDBC[18]. Odseparowanie poszczególnych warstw sprawia, że moduły są od siebie zależne w możliwie najmniejszym stopniu. W konsekwencji architektura pozwala na niezależny rozwój każdej części systemu, co znacząco przekłada się na przejrzystość rozwiązania. Skomplikowana logika systemu zaimplementowana została na serwerze, dzięki czemu aplikacje klienckie są lekkie i wysoce efektywne, a ruch sieciowy pomiędzy serwerem a bazą danych został znacząco ograniczony. Rozwiązanie umożliwia również łatwe wprowadzanie zmian w implementacji udostępnianych metod, bez konieczności aktualizowania aplikacji klienckich. Ważną cechą systemu jest możliwość działania aplikacji klienckich w środowiskach heterogenicznych.



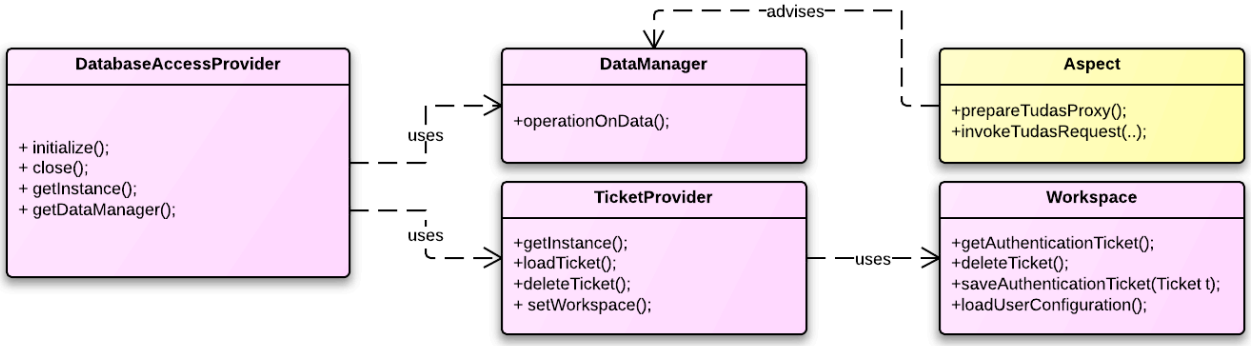
Rysunek 4: „Ogólna architektura systemu".

18 JDBC (ang. Java DataBase Connectivity) - interfejs programowania umożliwiający niezależnym od platformy aplikacjom napisanym w języku Java porozumiewać się z bazami danych za pomocą języka SQL.

5.2. Klient

Głównym zadaniem modułów klienckich jest udostępnienie usług oferowanych przez system. DatabaseAccessProvider umożliwia zainicjalizowanie/zamknięcie połączenia z serwerem TUDAS oraz pobranie grup metod oferujących konkretne usługi. Każdy z manager'ów (serwis'ów) danych udostępniają metody do zapisu i odczytu danych z bazy. Wywołania klienckie ma metodach manager'ów są delegowane do proxy ICE'a i przetwarzane na serwerze. Dodatkowo, aplikacja kliencka zapisuje istotne informacje w swoim workspace'ie, który jest tworzony podczas instalacji (patrz: "Dokumentacja użytkownika"). W trakcie pracy zapisywane logi z wywołań bibliotek TUDAS, , przechowywane są dane użytkownika i ticket, który jest niezbędny przy procesie autoryzacji. Metody serwisów (manager'ów) są otoczone aspektami co sprawia, że każdym wywołaniu metody wykonywany jest kod uwierzytelniający użytkownika. W kliencie napisanym w języku Java wspomniana funkcjonalność została zrealizowana za pomocą narzędzia AspectJ, a w pozostałych bibliotekach klienckich za pomocą specyficznych konstrukcji językowych. Szczegóły na temat systemu autoryzacji i uwierzytelniania zostały opisane w rozdziale 7.

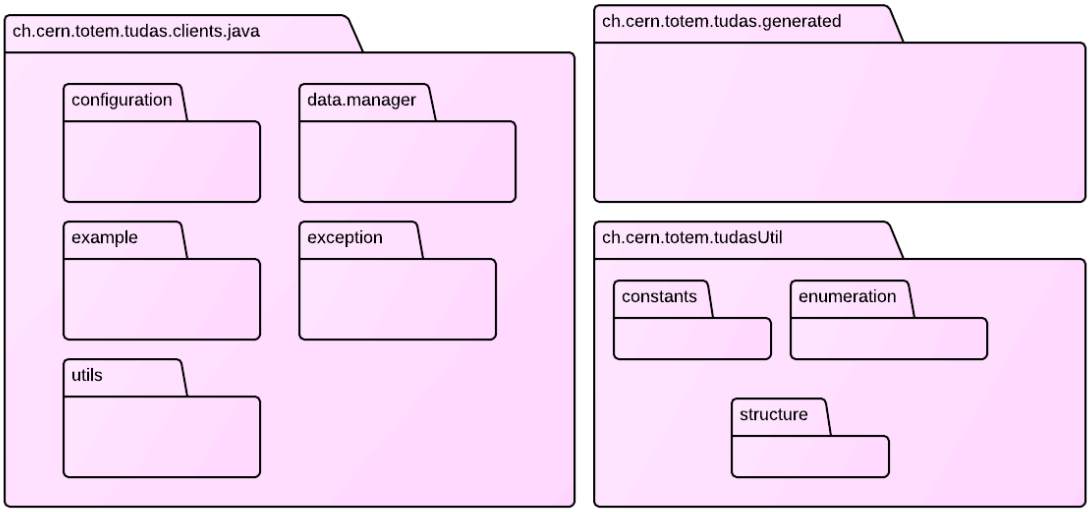
Biblioteki klienckie zostały stworzone dla trzech języków programowania: Java, C++ i Python - schemat rozwiązania jest w nich identyczny. Poniżej został przedstawiony diagram klas odpowiedzialnych za wspomniane mechanizmy:



Rysunek 5: „Ogólny diagram klas strony klienckiej".

5.2.1. Klient Java

Klasy klienta Javy zostały pogrupowane w odpowiednie pakiety, tak by odróżnić kod generowanych proxy od kodu klienckiego. Przeznaczenie poszczególnych pakietów zostało opisane poniżej.



Rysunek 6: „Diagram pakietów klienta Java".

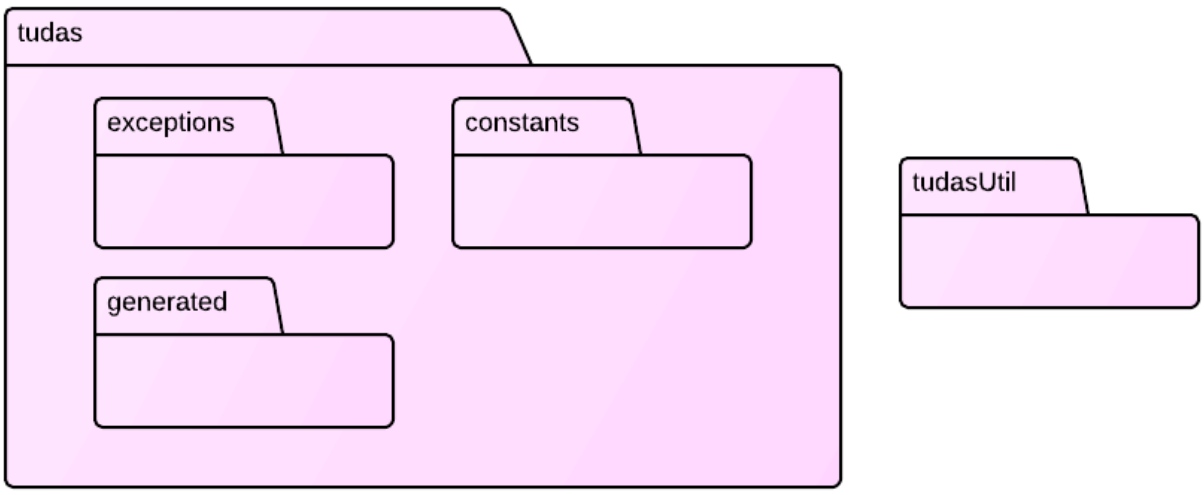
Nazwa pakietu	Zawartość
ch.cern.totem.tudas.clients.java	Głowny pakiet modułu klienta. Zawiera klasę DatabaseAccessProvider, dzięki której użytkownik zarządza managerami danych oraz inicjalizuje/zamyka połączenie z serwerem Tudas.
ch.cern.totem.tudas.clients.java.configuration	Klasy potrzebne do skonfigurowania workspace klienta oraz logika mechanizmów autoryzacji i uwierzytelniania.
ch.cern.totem.tudas.clients.java.data.manager	Zawiera managery danych udostępniające funkcje zapisu/odczytu danych. Opis interfejsów został zawarty w "Dokumentacji użytkownika".
ch.cern.totem.tudas.clients.java.example	Przykładowe aplikacje napisane w oparciu o bibliotekę kliencką. Zawiera przykłady użycia każdej metody, każdego managera danych.
ch.cern.totem.tudas.clients.java.exception	Wyjątki klienckie zawierające pełną informację o błędzie, jego przyczynie i proponowanym sposobie rozwiązania problemu.

ch.cern.totem.tudas.clients.java.utils	Stałe używane tylko i wyłącznie po stronie klienckiej.
ch.cern.totem.tudas.generated	Generowane klasy przez slice2java. Zawierają między innymi implementacje proxy.
ch.cern.totem.tudas.tudasUtil.constants	Stałe współdzielone pomiędzy klientem i serwerem.
ch.cern.totem.tudas.tudasUtil.enumeration	Typy wyliczeniowe współdzielone między klientem i serwerem.
ch.cern.totem.tudas.tudasUtil.structure	Struktury potrzebne do pakowania danych w paczki. Tego typu struktury są całościowo wysyłane do przetworzenia na serwerze.

Tabela 1: „Pakiety klienta Java".

5.2.2. Klient C++

Poniżej zostały przedstawione przestrzenie nazw klienta napisanego w języku C++.



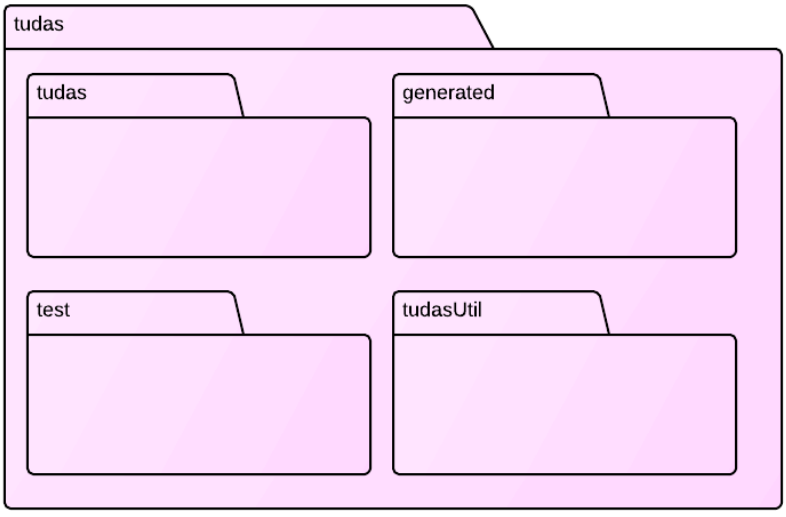
Rysunek 7: „Diagram pakietów klienta C++".

Nazwa pakietu	Zawartość
tudas	Główny pakiet biblioteki TUDAS. Zawiera między innymi klasę DatabaseAccessProvider, klasy odpowiedzialne za zarządzanie Workspace oraz mechanizmem autoryzacji i uwierzytelniania. Tutaj też znajdują się wszystkie klasy reprezentujące serwisy (managery) TUDAS.
tudas.exceptions	Wyjątki klienckie zawierające pełną informację o błędzie, jego przyczynie i proponowanym sposobie rozwiązania problemu.
tudas.constants	Stałe używane po stronie klienta.
tudas.generated	Wygenerowane klasy ICE'a zawierające m.in. implementację proxy.
tudasUtil	Stałe, typy wyliczeniowe i struktury współdzielone pomiędzy klientem i serwerem.

Tabela 2: „Diagram pakietów klienta C++".

5.2.3. Klient Python

Poniżej zostały przedstawione pakiety klienta napisanego w języku Python.



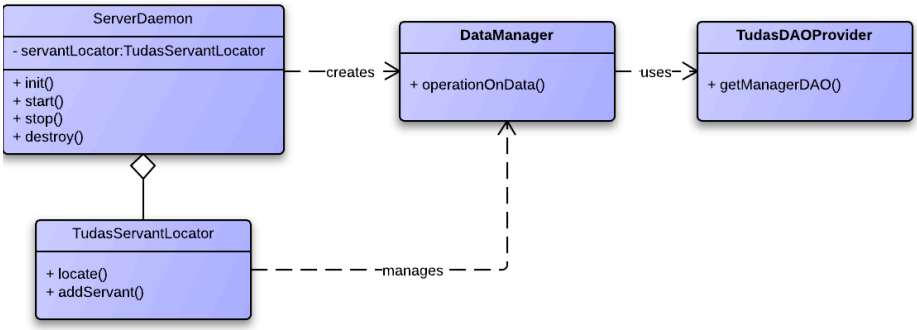
Rysunek 8: „Diagram pakietów klienta Python".

Pakiet	Zawartość
tudas	Główny pakiet biblioteki TUDAS. Zawiera między innymi moduł *core* - bazowy moduł biblioteki TUDAS. On z kolei zawiera klasę DatabaseAccessProvider.
tudas.tudas	Tutaj znajdują się wszystkie klasy reprezentujące serwisy (manager'y) TUDAS.
tudas.generated	Wygenerowane klasy ICE'a zawierające m.in. implementację proxy.
tudas.tudasUtil	Stałe, typy wyliczeniowe i struktury współdzielone pomiędzy klientem i serwerem.

Tabela 3 „Pakiety klienta Python".

5.3. Serwer

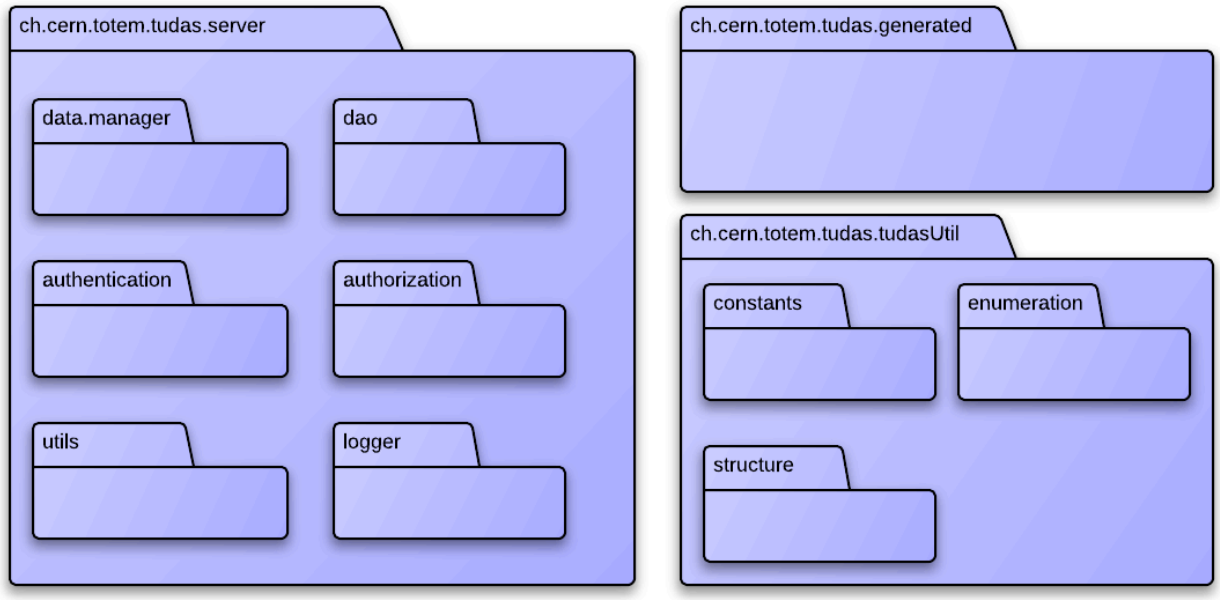
Zadaniem serwera jest przetwarzanie żądań klientów. Dzięki technologii ICE, każde zapytanie trafia do Servant Locatora, który po wstępnej weryfikacji (więcej w rozdziale 7) przekazuje wywołanie do odpowiedniego obiektu serwanta, zarejestrowanego w adapterze[19] podczas inicjalizacji serwera. Serwantami serwera TUDAS są implementacje usług (data managerów), które dokonują walidacji parametrów wywołania, a następnie odwołują się do obiektów dostępowych z warstwy database interfaces (wykorzystując wzorzec projektowy Database Access Object).



Rysunek 9: „Ogólny diagram najważniejszych komponentów części serwerowej".

Ogólna architektura serwera uwzględnia także warstwę autoryzacji i logowania (realizowane aspektowo) oraz mechanizm tworzenia i transmisji wyjątków. Wszystkie te zagadnienia zostały dokładniej opisane w dalszych rozdziałach.

19 Adapter - obiekt odpowiedzialny za przekazanie żądania do właściwego serwanta.



Rysunek 10: „Diagram pakietów części serwerowej".

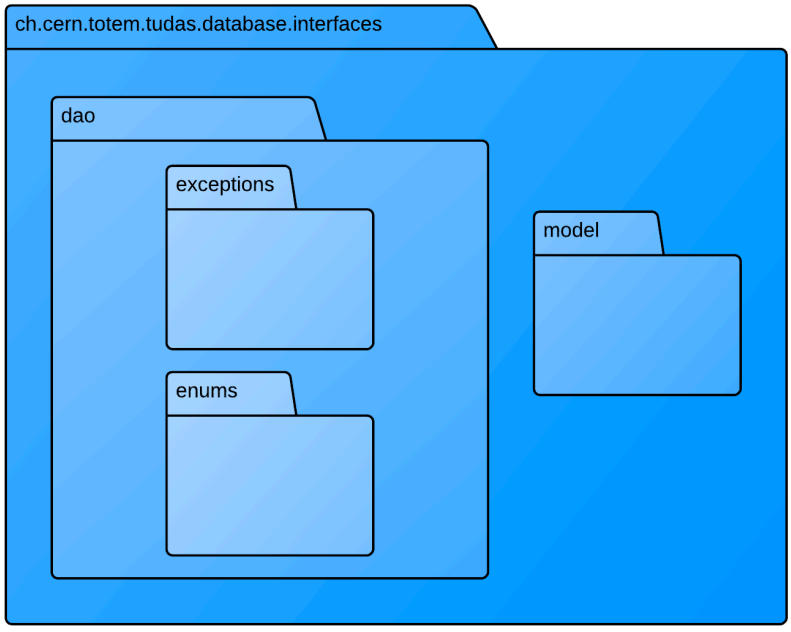
Nazwa pakietu	Zawartość
ch.cern.totem.tudas.server	Głowny pakiet modułu serwera. Zawiera podstawowe klasy uruchomieniowe serwera (proces, daemon)
ch.cern.totem.tudas.server.data.manager	Implementacje serwerowe interfejsów ICE dla wszystkich usług TUDAS
ch.cern.totem.tudas.server.authentication	Moduł uwierzytelniania, klasy zajmujące się walidacją użytkowników i generowaniem nowych tokenów (ticketów)
ch.cern.totem.tudas.server.authorization	Moduł autoryzacji, zawiera aspektową realizację funkcji sprawdzających uprawnienia użytkownika do wykonania danej operacji
ch.cern.totem.tudas.server.logger	Narzędzia związane z logami, zbieranymi po stronie serwera
ch.cern.totem.tudas.server.utils	Stałe i narzędzia używane tylko i wyłącznie po stronie serwera
ch.cern.totem.tudas.generated	Generowane klasy przez slice2java. Zawierają między innymi implementacje proxy

ch.cern.totem.tudas.tudasUtil.constants	Stałe współdzielone pomiędzy klientem i serwerem
ch.cern.totem.tudas.tudasUtil.enumeration	Typy wyliczeniowe współdzielone między klientem i serwerem
ch.cern.totem.tudas.tudasUtil.structure	Struktury potrzebne do pakowania danych w paczki. Tego typu struktury są całościowo wysyłane do przetworzenia na serwerze

Tabela 4: „Pakiety C++".

5.4. Database interfaces

Główną odpowiedzialnością stawianą przed tym modułem jest zdefiniowanie kontraktu pomiędzy modułami: serwerowym i bazodanowym. Takie rozwiązanie pozwala na niezależny rozwój każdego z wcześniej zdefiniowanych modułów. W razie potrzeby istnieje możliwość wymiany dowolnego z tych modułów na inny spełniający zapisany kontrakt. Na zawartość tej części systemu składają się głównie definicje interfejsów, deklaracje metod dokonujących zmian na bazie danych, definicje struktur danych oraz wyjątków rzucanych przez moduł bazodanowy.



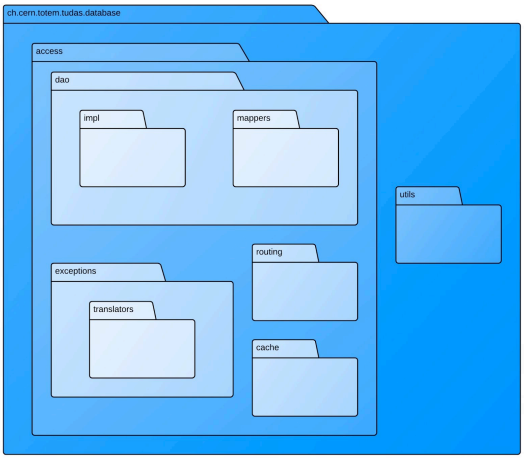
Rysunek 11: „Pakiety modułu DatabaseInterfaces"

Nazwa pakietu	Zawartość
ch.cern.totem.tudas.database.interfaces.dao	Interfejsy umożliwiające realizację określonych funkcjonalności na bazie danych
ch.cern.totem.tudas.database.interfaces.dao.enums	Stałe wyliczeniowe dla takich wartości jak: rodzaj uprawnień, typ pomiary
ch.cern.totem.tudas.database.interfaces.dao.exceptions	Wyjątki, które może zwrócić moduł database access do modułu serwerowego w przypadku błędy przetwarzania żądania
ch.cern.totem.tudas.database.interfaces.model	Obiektowy model danych

Tabela 5: „Pakiety modułu DatabaseInterfaces".

5.5. Database access

Jedno z wymagań niefunkcjonalnych mówiące o tym, że system ma współpracować z ściśle określonym systemem zarządzania bazą danych pozwoliło na uzależnienie modułu od bazy danych jednocześnie zyskując efektywność działania. Główną częścią modułu odpowiedzialnego za dostęp do bazy jest implementacja interfejsów zlokalizowanych w module za to odpowiedzialnym, konwertowanie wyników zapytań SQL-owych do zdefiniowanych struktur oraz tłumaczenie błędów bazodanowych na odpowiednie wyjątki.



Rysunek 12: „Pakiety modułu DatabaseAccess".

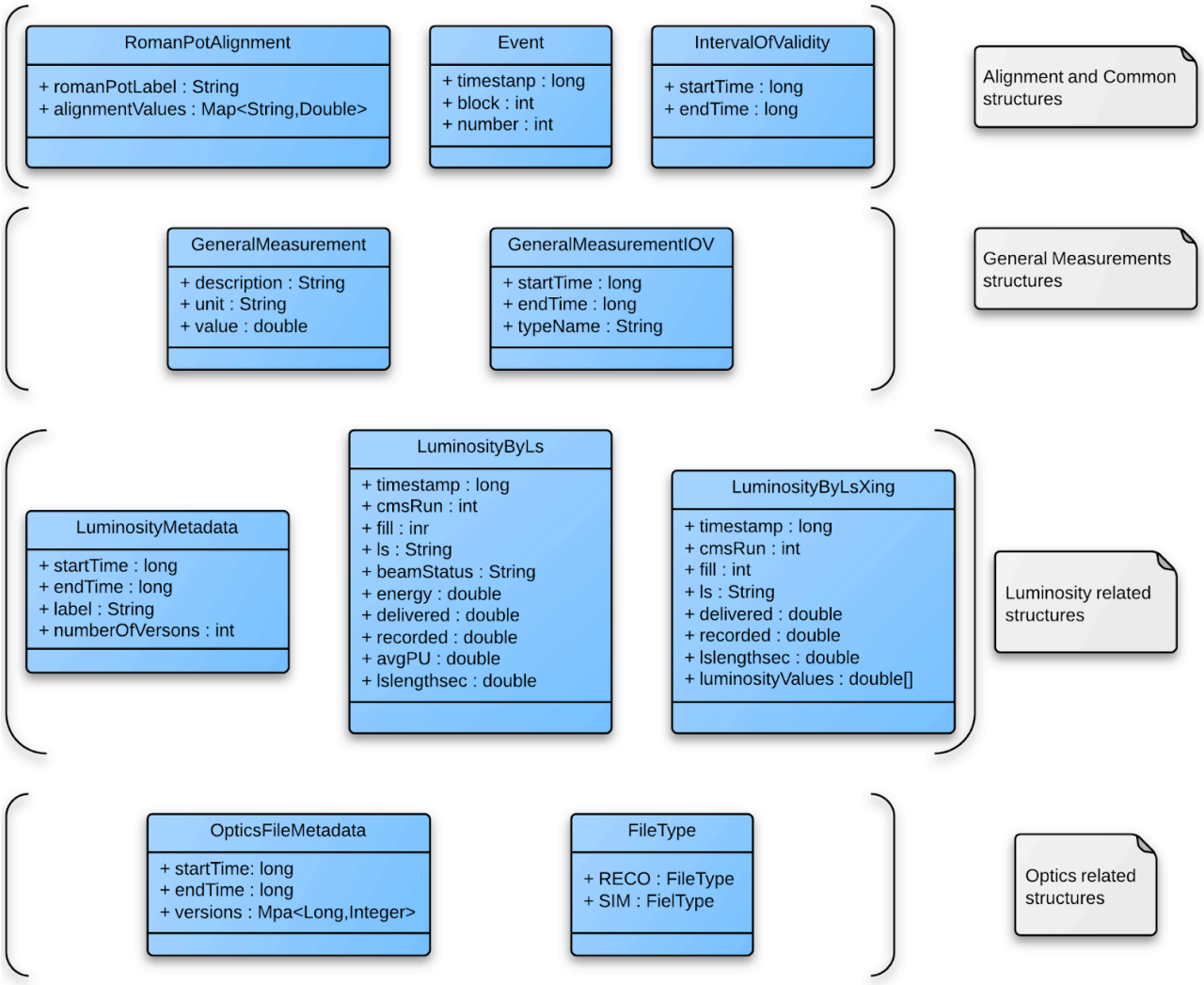
Nazwa pakietu	Zawartość
ch.cern.totem.tudas.database.access.cache	Zarządca mechanizmu cache oraz struktury dla kluczy obiektów przechowywanych w cache'u
ch.cern.totem.tudas.database.access.dao	Wewnętrzne moduły bazy danych
ch.cern.totem.tudas.database.access.dao.impl	Implementacja interfejsów DAO zarówno tych wewnętrznych jak i również znajdujących się w module interfejsów.
ch.cern.totem.tudas.database.access.dao.mappers	Klasy odpowiedzialne za mapowanie rezultatów komend sql na modele obiektowe danych
ch.cern.totem.tudas.database.access.exceptions	Grupa wyjątków rzucanych w razie błędów związanych z bazą danych
ch.cern.totem.tudas.database.access.exceptions.translators	Translator kodów błędów zwróconych przez bazę danych do odpowiedniego rodzaju wyjątku
ch.cern.totem.tudas.database.access.routing	Klasy realizujące dobór konta bazodanowego w zależności od przeprowadzanej operacji
ch.cern.totem.tudas.database.utils	Klasy pomocnicze takie jak: konwertery błędów, struktur, komendy sql-owe, stałe itp.

Tabela 6: „Pakiety modułu DatabaseAccess".

6. Model danych

6.1. Schemat obiektowy/strukturalny

Dla złożonych wartości pomiarów zostały zaprojektowane specjalne struktury agregujące potrzebne wartości. Tak zdefiniowane struktury ułatwiają korzystanie z aplikacji oraz zmniejszają ryzyko popełnienia błędu przez klienta wykorzystującego bibliotekę.



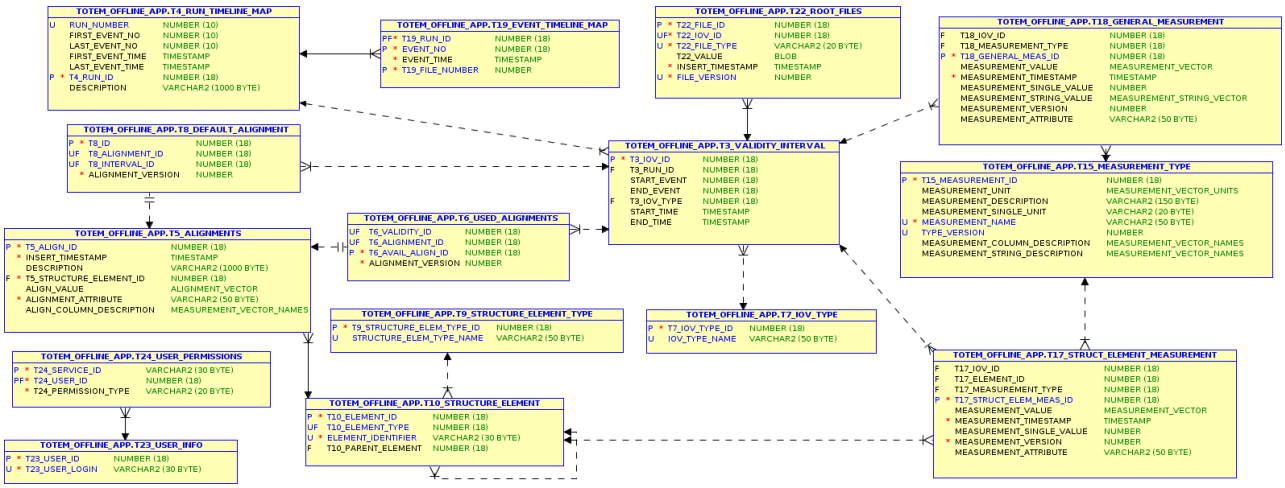
Rysunek 13 „Struktury danych wykorzystywane przez aplikacje klienckie oraz część serwerową".

Część bazodanowa ma analogiczne struktury jednak jej pola są prywatne z odpowiednimi możliwościami ich ustawiania oraz pobierania.

6.2. Schemat bazy danych

Baza danych eksperymentu Totem została zaprojektowana do przechowywania kilku rodzajów danych. W rezultacie na schemacie można zauważyć następujące, najważniejsze grupy tabel:

- tabele związane z ogólnymi wynikami pomiaru,
- tabele związane z wynikami pomiarów przeprowadzonych przez różne części detektorów,
- tabele przechowujące wyrównania ruchomych detektorów.



Rysunek 14: „Schemat bazy danych".

6.3. Opis schematu

Na schemacie zostały zaznaczone tylko tabele wykorzystywane przez system. Pełny schemat wraz z opisem można odnaleźć na stronach eksperymentu TOTEM.[20]

6.3.1. T3_VALIDITY_INTERVAL, T7_IOV_TYPE

Przedział ważności pomiaru stanowi centralny punkt schematu ponieważ umożliwia on wyszukiwanie aktualnych pomiarów w podanej chwili czasu. Dodatkowo jest on połączony z prawie wszystkimi tabelami w schemacie (pośrednio lub bezpośrednio z wyjątkiem tabel związanych z autoryzacją). T7_IOV_TYPE jest tabelą słownikową przechowującą typ pomiaru oraz ułatwiającą i przyspieszającą wyszukiwanie.

20 Pełny schemat bazy danych [https://twiki.cern.ch/twiki/bin/view/TOTEM/CompDBTudasDatabaseSchemaDescription]

Nazwa tabeli	T3_VALIDITY_INTERVAL	
Opis	Przechowuje czas ważności każdego z pomiarów	
Pola	t3_iov_id	liczba całkowita ,klucz główny, bez znaczenia domenowego
	t3_run_id	liczba całkowita, klucz obcy tabeli T4_RUN_TIMELINE_MAP
	start_event	liczba całkowita ,numer pierwszego zdarzenie w danym przedziale czasu
	end_event	liczba całkowita, numer ostatniego zdarzenia w danym przedziale czasu
	t3_iov_type	liczba całkowita ,klucz obcy tabeli T7_IOV_TYPE
	start_time	timestamp, czas rozpoczęcia przedziału ważności
	end_time	timestamp, czas zakończenia przedziału ważności
Relacje zewnętrzne	T3_VALIDITY_INTERVAL	wiele do jednego
	T8_DEFAULT_ALIGNMENTS	wiele do jednego
	T13_SENSOR_PART_STATUS	wiele do jednego
	T16_SENSOR_PART_MEASUREMENTS	wiele do jednego
	T17_STRUCT_ELEMENT_MEASUREMENT	wiele do jednego
	T18_GENERAL_MEASUREMENT	wiele do jednego
	T22_ROOT_FILES	wiele do jednego

Tabela 7: „Opis tabeli T3_VALIDITY_INTERVAL".

Nazwa tabeli	T7_IOV_TYPE	
Opis	Przechowuje informacje na temat typu przedziału ważności	
Pola	t7_iov_type_id	liczba całkowita, klucz główny tabeli, bez znaczenia domenowego
	iov_type_name	string, nazwa typu
Relacja zewnętrzna	T3_VALIDITY_INTERVAL	wiele do jednego

Tabela 8: „Opis tabeli T7_IOV_TYPE".

6.3.2. T15_MEASUREMENT_TYPE

Tabela przechowująca typ pomiaru jest tabelą słownikową zawierającą nie tylko nazwę pomiaru, ale także jego jednostkę oraz znaczenie[21]. Dodatkowe pole type_version umożliwia przechowywanie wielu wersji typów różniących się znaczeniem lub ilością wartości. Dla przykładu pomiar wielkości X wartości o nazwach a, b oraz c. Po pewnym czasie nastąpiła modyfikacja czujnika wykonującego ten pomiar i teraz zwraca on dodatkowo dla wielkości X wartość d. W takiej sytuacji zostanie stworzona nowa wersja typu pomiaru wielkości X.

Nazwa tabeli	T15_MEASUREMENT_TYPE	
Opis	Przechowuje informację na temat dostępnych typów pomiaru. Przechowuje także nazwy wartości oraz jednostki (głównie dla tablic pomiarów)	
Pola	t15_measurement_id	liczba całkowita, klucz główny, bez znaczenia domenowego
	measurement_unit	tablica stringów, jednostki dla każdej wielkości w przypadku gdy pomiar składa się z więcej niż jednej wielkości
	measurement_column_description	tablica stringów, nazwy wielkości w przypadku gdy pomiar składa się z więcej niż jednej wielkości

21 Znaczenie pomiaru - część pomiarów składa się z kilku wielkości, dlatego ważne jest aby rozróżnić która wartość należy do której wielkości

	measurement_descriptio n	string, nazwa wielkości w przypadku gdy pomiar składa się z pojedyńczej wartości
	measurement_single_unit	string, jednostka pomiaru w przypadku gdy pomiar składa się z pojedyńczej wartości
	measurement_name	string, nazwa typu pomiaru
	measurement_string_des cription	tablica stringów, nazwy wielkości dla pomiarów których wartości są wyrażone w postaci stringów
	type_version	liczba całkowita, numer wersji typu pomiaru
Relacje zewnętrzne	T17_STRUCT_ELEMENT _MEASUREMENT	wiele do jednego
	T18_GENERAL_MEASU REMENT	wiele do jednego

Tabela 9: „Opis tabeli T15_MEASUREMENT_TYPE".

6.3.3. T18_GENERAL_MEASUREMENT

Przechowuje ogólne date dotyczące warunków panujących w tunelu - nie związane z żadną częścią detektora. Pole measurement_version umożliwia przechowywanie kilku wersji pomiaru dla tego samego przedziału ważności. Dla przykładu Obliczając wartości optyki została użyta pewna stała, która sprawiła, że dokładność obliczeń wyniosła 0.1 %. Po pewnym czasie dla tych samych danych wejściowych (z wyjątkiem stałej) użyto innej wartości i otrzymano dokładniejszy rezultat. Wspomniane pole umożliwia zapamiętanie tych dwóch wyników. Pole measurement_version zapewnia możliwość przechowywanie dodatkowej wartości opisującej pomiar np. dla świetlności jest to version key[22].

Nazwa tabeli	**T18_GENERAL_MEASUREMENT**	
Opis	Przechowuje ogólne dane na temat warunków panujących w tunelu	
Pola	t18_general_meas_id	liczba całkowita, klucz główny, bez znaczenia domenowego

22 ang. Version Key - zestaw parametrów identyfikujących pomiar świetlności.

t18_iov_id	liczba całkowita, klucz obcy tabeli T3_VALIDITY_INTERVAL
t18_measurement_type	liczba całkowita, klucz obcy tabeli T15_MEASUREMENT_TYPE
measurement_value	tablica liczb, wyniki pomiarów wielkości w przypadku gdy pomiar składa się z więcej niż jednej wielkości
measurement_string_value	tablica stringów, wyniki pomiarów wielkości, które wyrażone są w postaci stringów.
measurement_single_value	liczba, wynik pomiaru, gdy składa się on z pojedyńczej wielkości
measurement_timestamp	timestamp, czas pobrania pomiaru
measurement_attribute	string, dodatkowa informacja opisująca pomiar
measurement_version	liczba całkowita, numer wersji pomiaru

Tabela 10: „Opis tabeli T18_GENERAL_MEASUREMENT".

6.3.4. T10_STRUCT_ELEMENT, T9_STRUCTTURE_ELEMENT_TYPE

Każda z tych dwóch tabel pozwala odzwierciedlić rzeczywistą budowę systemu detektorów w bazie danych. Pierwsza z nich przechowuje dane pomagające w mapowaniu rzeczywistych części detektora na wirtualne identyfikatory tych części. Druga z tabel słownikowych (T9_STRUCTURE_ELEMENT_TYPE) zawiera nazwę części detektora np. VFAT[23].

Nazwa tabeli	**T10_STRUCTURE_ELEMENT**	
Opis	Przechowuje dane na temat budowy systemu detektorów w eksperymencie	
Pola	t10_element_id	liczba całkowita, klucz główny, bez znaczenia domenowego

23 VFAT (ang Very Forward ATLAS and TOTEM chip) - układ scalony detektora odpowiedzialny za zbieranie danych z eksperymentu.

	t10_element_type	liczba całkowita, klucz obcy tabeli T9_STRUCTURE_ELEMENT_TYPE
	element_identifier	string, nazwa elementu umożliwiająca mapowanie rzeczywistych częśći detektora na wirtualne identyfikatory
	t10_parent_element	liczba całkowita, klucz główny do rodzica tej części w hierarchicznej budowie detektora
Relacje zewnętrzne	T5_ALIGNMENTS	wiele do jednego
	T10_STRUCTURE_ELEMENT	jeden do jednego
	T17_STRUCT_ELEMENT_MEASUREMENT	wiele do jednego

Tabela 11: „Opis tabeli T10_STRUCTURE_ELEMENT".

Nazwa tabeli	**T9_STRUCTURE_ELEMENT_TYPE**	
Opis	Przechowuje dane na temat rodzajów części systemu detektorów	
Pola	t9_structure_element_type_id	liczba całkowita, klucz główny, bez znaczenia domenowego
	structure_element_type_name	string, nazwa typu odzwierciedlająca rodzaj części detektora
Relacje zewnętrzne	T10_STRUCTURE_ELEMENT	wiele do jednego

Tabela 12: „Opis tabeli T9_STRUCTURE_ELEMENT_TYPE".

6.3.5. T17_STRUCT_ELEMENT_MEASUREMENT

Dana tabela zawiera wyniki pomiarów związanych z częściami systemu detektorów. Pole measurement_version daje możliwość wersjonowania danych, natomiast measuremt_attribute pomaga w przechowywaniu dodatkowego atrybutu opisującego pomiar.

Nazwa tabeli	T1_STRUCT_ELEMENT_MEASUREMENT	
Opis	Przechowuje dane pomiarów związanych częścią systemu detektorów	
Pola	t17_struct_elem_meas_id	liczba całkowita, klucz główny, bez znaczenia domenowego
	t17_element_id	liczba całkowita, klucz obcy T10_STRUCTURE_ELEMENT
	t17_iov_id	liczba całkowita, klucz obcy tabeli T3_VALIDITY_INTERVAL
	t17_measurement_type	liczba całkowita, klucz obcy tabeli T15_MEASUREMENT_TYPE
	measurement_value	tablica liczb, wyniki pomiarów wielkości w przypadku gdy pomiar składa się z więcej niż jednej wielkości
	measurement_single_value	liczba, przechowuje wynik pomiaru, gdy składa się on z pojedńczej wielkości
	measurement_timestamp	timestamp, czas pobrania pomiaru
	measurement_attribute	string, dodatkowa informacja opisująca pomiar
	measurement_version	liczba całkowita, numer wersji pomiaru

Tabela 13: „Opis tabeli T1_STRUCTURE_ELEMENT_MEASUREMENT".

6.3.6. T5_ALIGNMENTS, T8_DEFAULT_ALIGNMENT, T6_USED_ALIGNMENTS

Wszystkie z wymienionych tabel przechowują informację dotyczące wyrównań detektorów typu Roman Pot. Pierwsza z nich przechowuje dane pomiarów dla części detektora. Pole

alignmente_attribute umożliwia zapamiętywanie dodatkowej specyficznej wielkości charakteryzującej pomiar. T8_DEFAULT_ALIGNMENT zawieraja najnowszą wersję wyrównań części detektorów, w zależności od przedziału czasu ważności, natomiast ostatnia z tabel umożliwia wersjonowanie danych.

Nazwa tabeli	**T5_ALIGNMENTS**	
Opis	Przechowuje dane wyrównań części detektorów	
Pola	t5_align_id	liczba całkowita, klucz główny, bez znaczenia domenowego
	t5_element_id	liczba całkowita, klucz obcy T10_STRUCTURE_ELEMENT
	align_value	tablica liczb, wyniki pomiarów wyrównań pojedyńczej części detektora
	align_column_descrption	tablica stringów, nazwa każdej wielkości wyrówanania
	insert_timestamp	time, czas zapisu pomiarów
	description	string, opis/komentarz do wyników
	alignment_attribute	string, dodatkowa specyficzna informacja opisująca pomiar
Relacje zewnętrzne	T6_USED_ALIGNMENTS	jeden do jednego
	T8_DEFAULT_ALIGNMENT	jeden do jednego

Tabela 14: „Opis tabeli T5_ALGINMENTS".

Nazwa tabeli	**T8_DEFAULT_ALIGNMENT**
Opis	Łączy ostatnią wersję pomiarów w danym przedziale ważności do przedziału ważności danych

Pola	t8_id	liczba całkowita, klucz główny, bez znaczenia domenowego
	t8_alignment_id	liczba całkowita, klucz obcy T5_ALIGNMENTS
	t8_interval_id	liczba całkowita, klucz obcy tabeli T3_VALIDITY_INTERVAL
	alignment_version	liczba całkowita, numer wersji pomiaru

Tabela 15 „Opis tabeli T8_DEFAULT_ALIGNMENT"

Nazwa tabeli	**T6_USED_ALIGNMENTS**	
Opis	Łączy poprzednie wersje pomiarów do przedziału ważności danych	
Pola	t6_avail_align_id	liczba całkowita, klucz główny, bez znaczenia domenowego
	t6_alignment_id	liczba całkowita, klucz obcy T5_ALIGNMENTS
	t6_validity_id	liczba całkowita, klucz obcy tabeli T3_VALIDITY_INTERVALdetektora
	alignment_version	liczba całkowita, informacja o numerze wersji pomiaru

Tabela 16: „Opis tabeli T6_USED_ALIGNMENTS"

6.3.7. T4_RUN_TIMELINE_MAP, T19_EVENT_TIMELINE_MAP

Tabele zawierają kolejno podstawowe informacje dotyczące run'ów CMS oraz event'ów[24].

Nazwa tabeli	**T4_RUN_TIMELINE_MAP**	
Opis	Zawiera informacje dotyczące run'ów	
Pola	t4_run_id	liczba całkowita, klucz główny, bez znaczenia domenowego

24. ang. Event - zbiór surowych danych z detektorów pobranych w bardzo krótkim odstępie czasu np w chwili zderzenia 2 cząstek.

	run_number	liczba całkowita, umożliwia mapowanie pomiędzy wirtualnym identyfikatorem a rzeczywistym wydarzeniem
	first_event_no	liczba całkowita, numer pierwszego zdarzenia w run'ie
	last_event_no	liczba całkowita, numer ostatniego zdarzenia w run'ie
	first_event_time	timestamp, czas wystąpienia pierwszego zdarzenia w run'ie
	last_event_time	timestamp, czas wystąpienia ostatniego zdarzenia w run'ie
	description	string, dodatkowy opis run'u
Relacje zewnętrzne	T3_VALIDITY_INTERVAL	wiele do jednego
	T19_EVENT_TIMELINE_MAP	wiele do jednego

Tabela 17: „Opis tabeli T4_RUN_TIMELINE_MAP".

Nazwa tabeli	**T19_EVENT_TIMELINE_MAP**	
Opis	Zawiera informacje dotyczące event'ów	
Pola	t19_run_id	liczba całkowita, klucz obcy tabeli 4_RUN_TIMELINE_MAP, wchodzi w skład klucza głównego
	event_no	liczba całkowita, numer eventu (numer CMS), wchodzi w skład klucza głównego
	event_time	timestamp, czas wystąpienia event'u, wchodzi w skład klucza głównego
	t19_file_number	liczba całkowita, numer pliku w run'ie, wchodzi w skład klucza głównego

Tabela 18: „Opis tabeli T19_EVENT_TIMELINE_MAP".

6.3.8. T22_ROOT_FILES

Tabela służy do przechowywania plików optyki.

Nazwa tabeli	T22_ROOT_FILES	
Opis	Przechowuje pliki optyki nieelastycznej (inelastic)	
Pola	t22_file_id	liczba całkowita, klucz główny, bez znaczenia domenowego
	t22_iov_id	liczba całkowita, klucz obcy tabeli T3_VALIDITY_INTERVAL
	t22_file_type	string, typ pliku
	t22_value	blob, zawartość pliku
	insert_timestamp	time, czas zapisu pliku
	file_version	liczba całkowita, numer wersji pliku

Tabela 19: „Opis tabeli T22_ROOT_FILES".

6.3.9. T23_USER_INFO, T24_USER_PERMISSIONS

Dodatkowe tabele przechowujące prawa użytkowników systemu

Nazwa tabeli	T23_USER_INFO	
Opis	Przechowuje liste użytkowników systemu	
Pola	t23_user_id	liczba całkowita, klucz główny, bez znaczenia domenowego
	t23_user_login	string, login użytkownika
Relacja zewnętrzna	T24_USER_PERMISSIONS	wiele do jednego

Tabela 20: „Opis tabeli T23_USER_INFO".

Nazwa tabeli	T24_USER_PERMISSIONS	
Opis	Przechowuje prawa użytkowników do użytkownaia usług	
Pola	t24_user_id	liczba całkowita, klucz obcy tabeli T23_USER_INFO, wchodzi w skład klucza głównego
	t24_service_id	string, nazwa usługi, wchodzi w skład klucza głównego
	t24_permissions_type	string, rodzaj prawa dostępu do usługi

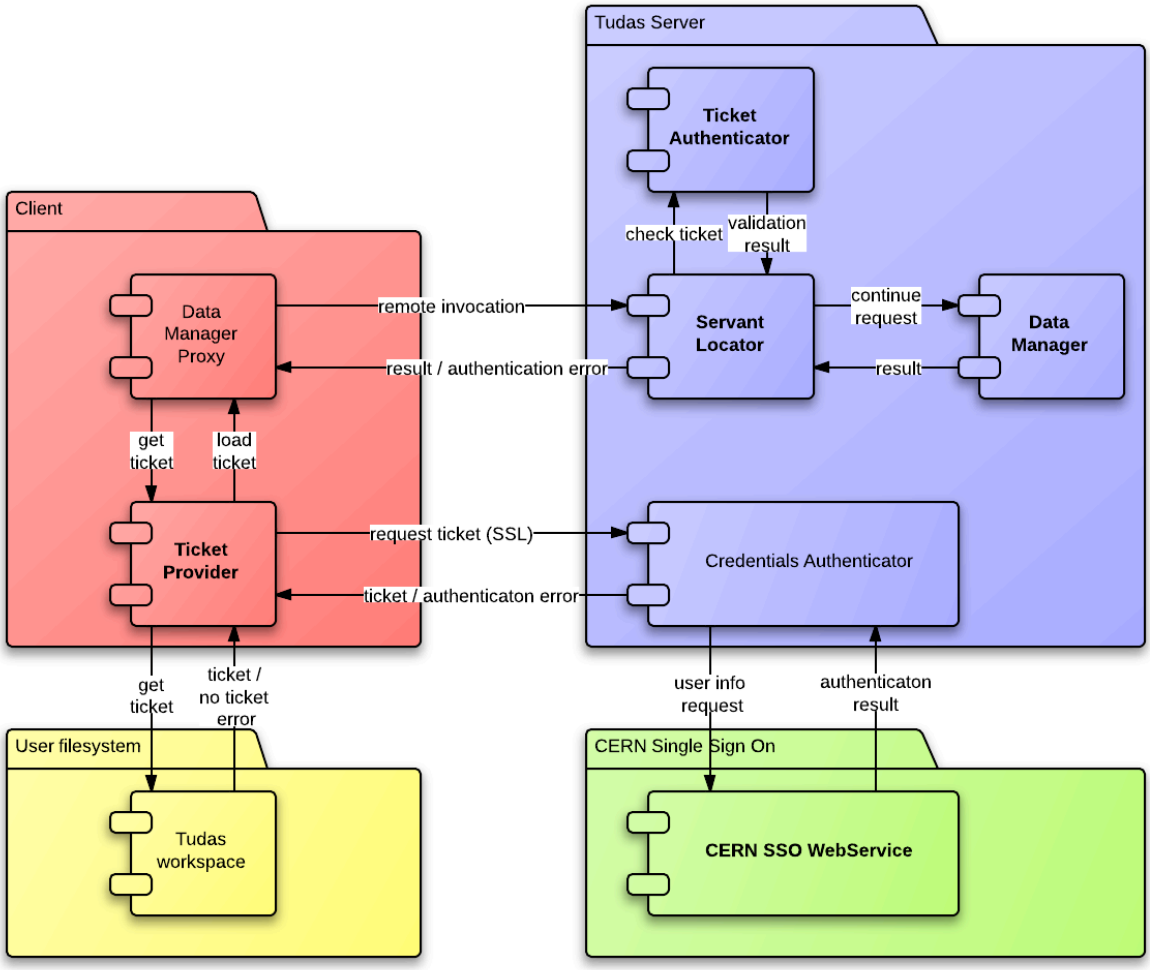
Tabela 21: „Opis tabeli T24_USER_PERMISSIONS".

7. Bezpieczeństwo

Dane z eksperymentów wymagają zabezpieczenia przed nieautoryzowanym dostępem. W tym celu system dostarcza złożony mechanizm uwierzytelniania i autoryzacji, opartego na koncepcji przyznawania tokenów (ticketów), upoważniających do korzystania z systemu przez określony czas. Transparentność wprowadzanych rozwiązań sprawia, że użytkownik może nawet nie być świadomy podejmowanych działań autoryzacyjnych i uwierzytelniających.

7.1. Uwierzytelnianie

Mechanizmy uwierzytelniania zostały zaimplementowane zarówno po stronie serwera, jak i w każdym z klientów (z uwzględnieniem specyfiki danego języka programowania).



Rysunek 14 „Mechanizm uwierzytelniania w systemie TUDAS".

Tożsamość każdego z użytkowników weryfikowana jest przy użyciu systemu CERN Single Sign On, dlatego warunkiem koniecznym użytkowania systemu jest posiadanie aktywnego konto w wspomnianym systemie. Potrzebne dane (login, hasło) winny zostać zapisane w przestrzeni roboczej (workspace'ie) klienta, zgodnie z instrukcjami zawartymi w przewodniku użytkownika.
Zgodnie z koncepcją ticketów, wraz z każdym zapytaniem, do serwera przesyłany jest zaszyfrowany ticket, zapisywany w workspace'ie. W momencie pierwszego uruchomienia aplikacji, korzystającej z systemu TUDAS (lub w każdej innej sytuacji, gdy ticket nie został znaleziony w workspace'ie), pierwsze wywołanie zdalnej metody automatycznie połączy się ze specjalnym zdalnym obiektem na serwerze, który zajmuje się dostarczaniem nowych ticketów. Jest to jedyna sytuacja, gdy system wykorzystuje szyfrowane połączenia SSL (bez certyfikatów). Moduł Credentials Authenticator korzysta z przesłanych danych (loginu i hasła odczytanych z workspace'a), delegując zapytanie do webservice'u CERN Single Sign On. W przypadku pozytywnego rezultatu uwierzytelniania w CERN SSO generowany jest nowy ticket, w którym zapisuje się dane o użytkowniku oraz datę ważności ticketu. Zaszyfrowany akutalnym kluczem serwera ticket zwracany jest do klienta, a następnie zapisywany w workspace'ie.
W przypadku wszystkich pozostałych wywołań (ticket znajduje się workspace'ie), ticket jest przesyłany w kontekście wywołania zdalnej metody, a następnie sprawdzany przed oddelegowaniem zapytania do konkretnego serwanta. Servant Locator stanowi w tym przypadku firewall, który przepuszcza jedynie poprawne i ważne tickety.
W przypadku otrzymania niepoprawnego (lub nieważnego) ticketu, zwracany jest odpowiedni błąd, który zostaje obsłużony po stronie klienta w sposób dla niego transparentny. Dokładna metoda obsługi tej sytuacji zależna jest od języka programowania strony klienckiej:

7.1.1. Java

Każde wywołanie na obiekcie proxy obiektu wygenerowanego przez *slice2java*, otoczone jest kodem w realizacji aspektowej. W momencie otrzymania błędu walidacji ticketa, klient usuwa stary ticket (również fizycznie, z workspace) i odwołuje się do Ticket Providera, który wykonuje czynności związane z uzyskaniem nowego ticketa. Potem następuje próba ponownego wywołania zdalnej metody i cały proces się powtarza.

7.1.2. Python

Każda z metod proxy obiektu wygenerowanego przez *slice2py* zostaje umieszczona w dekoratorze, który dba o ewentualne pobranie nowego ticketa i ponowienie zapytania w sytuacji błędu walidacji ticketa w dokładnie taki sposób, jak w przypadku języka Java.

7.1.3. C++

Wskaźnik do każdej metody proxy obiektu wygenerowanego przez slice2cpp zostaje przekazany do specjalnego obiektu, realizującego czynności opisane w powyższych przypadkach przy użyciu mechanizmu *variadic templates*, jednego z nowych elementów języka c++11.

7.2. Autoryzacja

Dla każdego z serwisów systemu TUDAS został zdefiniowany zestaw reguł, uprawniających do przeprowadzania operacji bazodanowych. Przed skorzystaniem z któregokolwiek z serwisów, użytkownik musi zgłosić się do administratora systemu, aby ten nadał mu odpowiednie uprawnienia. Proces ten odbywa się przy pomocy usług systemu - jednym z oferowanych serwisów jest Administration Manager, który pozwala na dodanie lub usunięcie uprawnień dla danego użytkownika (korzystanie z wspomnianego serwisu wymaga uprawnień administratora).
Dane o użytkownikach i ich serwisach przechowywane są również w TOTEM Offline Database, a przed każdym odwołaniem się do serwisu (już po przejściu przez proces uwierzytelniania i zwróceniu obiektu serwanta przez TUDAS Servant Locator'a), wywoływany jest aspektowy kod, sprawdzający dane użytkownika zapisane w tickecie. W celu zwiększania wydajności, wykorzystywany jest tu mechanizm pamięci podręcznej po stronie serwera, który umożliwia sprawdzenie uprawnień bez konieczności odwoływania się do bazy danych przy każdym wywołaniu.

7.3. Bezpieczeństwo danych

Moduł bazodanowy wyposażony jest w 2 mechanizmy bezpieczeństwa:
- dynamiczny dobór kont (routing bazodanowy),
- prepared statements[25].

Pierwszy mechanizm polega na wykorzystaniu dwóch kont z różnymi prawami wykonywania operacji. Pierwsze z nich ma tylko i wyłącznie prawa do odczytu zawartości bazy danych, natomiast drugie dodatkowo umożliwia wykonywanie operacji zmieniających zawartość bazy. W trakcie wykonywania dowolnej operacji bazodanowej wybierane jest konto z minimalnymi ale wystarczającymi prawami. W tej sytuacji wszystkie polecenia odczytu kierowane są do bazy poprzez konto tylko do odczyty co

25 ang. Prepared Statements - funkcjonalność użyta do wielokrotnego wykonywania tego samego zapytania z różnymi wartościami parametrów z dużą wydajnością

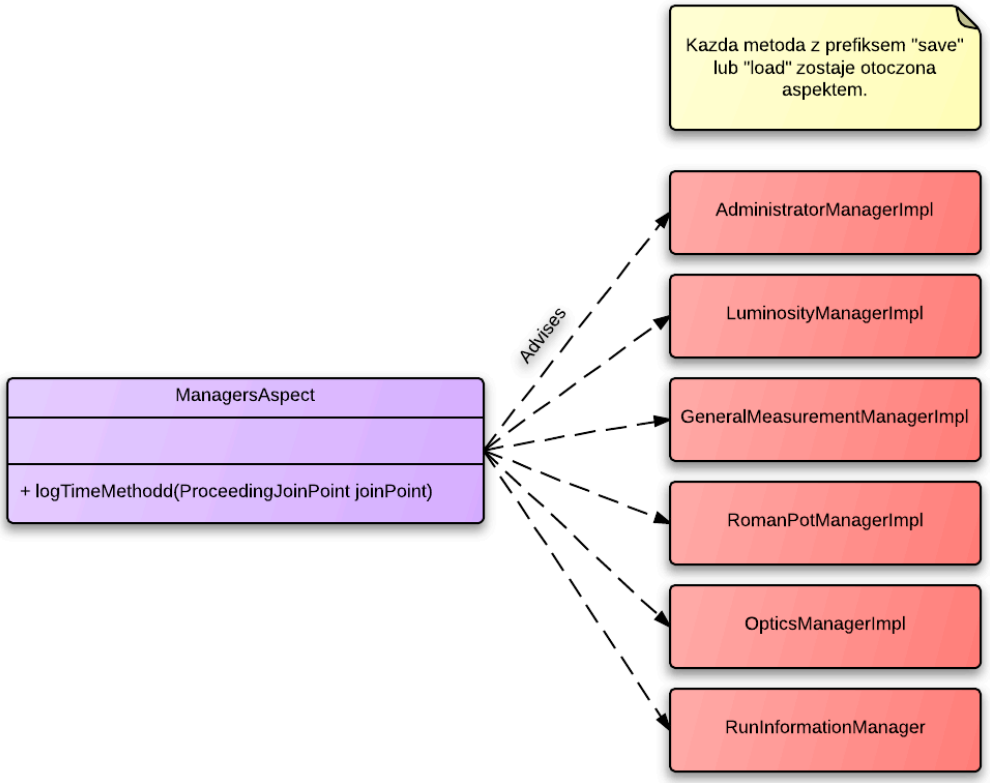
gwarantuje, że osoba posiadająca prawa tylko do odczytu nie będzie w stanie w żaden sposób (korzystając z systemu) zmienić zawartości bazy danych.

Część parametrów metod przesyłanych jest w postaci tekstowej dlatego też zaistaniała potrzeba zabezpieczenia systemu przed atakami typu sql injection[26]. Drugi mechanizm - prepared statement - gwarantuje bezpieczeństwo przy atakach tego typu.

26. ang. SQL injection - rodzaj ataku polegający na wykorzystaniu błędnego (niewystarczalnego) typowania parametrów zapytań bazodanowych.

8. Logowanie aktywności użytkowników

Jednym z wymagań użytkowników, był zapis aktywności użytkowników do pliku. Logowane powinny być podstawowe informacje na temat użytkownika wykonującego akcję, sygnatura wywoływanej metody oraz czas wykonywania metody na serwerze.



Rysunek 15: „Otoczenie aspektami metod managerów po stronie serwera."

Metody wewnątrz managerów danych nie mają świadomości istnienia aspektu. Użyte rozwiązanie - AspectJ pozwala, aby przy kompilacji kodu serwera otoczyć metody managerów odpowiednią metodą aspektu. Otaczane są jedynie metody których sygnatury mają prefix "load" lub "save" i należą do odpowiedniego pakietu (ch.cern.totem.tudas.server.data.manager). Dzięki użytemu rozwiązaniu, deweloper zostaje zwolniony z obowiązku rejestracji metod, lub pamiętaniu o wywołaniu odpowiednich metod logujących. Metoda aspektu pozwala na zapisanie informacji o aktywności użytkownika do pliku log/accounting.log przy pomocy loggera używanego w projekcie, udostępnionego przez bibliotekę log4j.

Przyjęty mechanizm pozwala na analizowanie typów danych jakie są najczęściej używane przez użytkowników oraz dostosowywanie implementacji, tak aby zapytania były efektywniej.

9. Mechanizm wyjątków

Proces formowania wyjątków w systemie TUDAS jest wielowarstwowy, szczelnie pokrywający całą dziedzinę potencjalnych sytuacji wystąpienia błędu. Głównym celem tej części systemu jest możliwość zaprezentowania użytkownikowi końcowemu prostej i czytelnej wiadomości, przy jednoczesnym utrzymaniu w logach szczegółowych informacji, przydatnych dla administratora i deweloperów. Rezultaty widoczne po stronie klienta zostały opisane w podręczniku użytkownika. Dany rozdział poświęcony jest mechanizmowi wyjątków z puntu widzenia dewelopera systemu.

9.1. Ogólna hierarchia wyjątków

Wyjątki w systemie Tudas podzielone są na dwie główne grupy:

- **Tudas ICE exceptions** - zdefiniowane w Slice, wszystkie dziedziczą po *TudasICEException* (lub bardziej precyzjnie: *TudasServerException* or *TudasClientException*). Są używane do przenoszenia informacji o wyjątku w warstwie komunikacji (od serwera do klienta) i powinny zostać przekonwertowane za pomocą *ExceptionConverter* na *TudasException*, zanim zostaną dostarczone do użytkownika końcowego.
- **Wyjątki użytkownika końcowego** - zaimplementowane w klasyczny sposób, wszystkie dziedziczą po*TudasException*. Powinny zawierać pełną informację o wyjątku (informacja, przyczyna, rozwiązanie) i mogą być zarządzane przez użytkownika końcowego.

9.2. Tworzenie nowych wyjątków

W zależności od potrzeb, istnieje kilka różnych możliwości na stworzenie nowych rodzajów wyjątków:

Rodzaj wyjątku	Projekt	Pakiet/ przestrzeń nazw	Dodatkowe informacje
Bazodanowy	database Interfaces	ch.cern.totem.tudas.database.interfaces.dao.exceptions	Należy rozszerzyć klasę ch.cern.totem.tudas.interfaces.dao.exceptions.DatabaseException i dodać konstrukcję if-else w ch.cern.totem.tudas.server.utils.ServerErrorConverter (projekt: server).

Serwerowy	server	ch.cern.totem.tudas.generated.exceptions (wygenerowany z SIICE)	Należy zdefiniować wyjątek w *slice/exceptions.ICE* i rozszerzyć klasę *TudasServerException*. Wyjątek powinien być rzucony jedynie po stronie serwerowej (jest przeznaczony do współpracy z protokołem ICE).
Kliencki	javaClient	ch.cern.totem.tudas.generated.exceptions (wygenerowany z SIICE)	Należy zdefiniować wyjątek w *slice*/exceptions.ICE i rozszerzyć klasę TudasClientException. Wyjątek powinien być przekształcony do TudasException za pomocą ch.cern.totem.tudas.java.clients.java.exception.ExceptionConverter.
Kliencki	python Client	tudas.generated.exceptions (wygenerowany z SIICE)	Należy zdefiniować wyjątek w *slice/exceptions.ICE* i rozszerzyć klasę *TudasClientException*. Wyjątek powinien być przekształcony do *TudasException* za pomocą *tudas.exception.ExceptionConverter*
Kliencki	cppClient	tudas::generated::exceptions (wygenerowany z SIICE)	Należy zdefiniować wyjątek w *slice/exceptions.ICE* i rozszerzyć klasę *TudasClientException*. Wyjątek powinien być przekształcony do *TudasException* za pomocą *tudas::exceptions::ExceptionConverter*

Tabela 22: „Rodzaj wyjątku".

9.3. Wyjątki bazodanowe

Obsługa wyjątków bazodanowych składa się z 2 etapów:

- walidacji po stronie systemu bazy danych oraz ewentualne zwrócenie kodu błędu
- walidacja po stronie serwerowej w module odpowiedzialnym za komunikację z bazą danych wraz z możliwym rzuceniem wyjątku.

9.3.1. Kody błędów

W pierwszym przypadku baza danych zwraca kod błędu. Może to być kod błędy zapisany w wyzwalaczu zdefiniowanym przez developera lub też kod błędu zarazerwowany przez system zarządzania bazą danych. W każdym z wspomnianych przypadków dochodzi do tłumaczenia kodu błędu na odpowiadający mu obiekt zdefiniowanej klasy wyjątku, jednak w różnych miejscach.
Jeżeli jest to kod błędu z wyzwalacza zdefiniowanego przez programistę przed rzuceniem błędu typu SqlException[27] sterowanie dochodzi do metody, która porównuje możliwe/przewidziane przez programistów kody błędów z aktualnym. W przypadku dopasowania kodów rzucany jest odpowiedni wyjątek i zostaje on obsłużony w module bazodanowym.
W drugim przypadku (brak dopasowania, kod błędu zarezerwowany dla systemu bazodanowego) gdochodzi do rzucenia wyjątku typu SqlException z zawartym w sobie kodzie błędu.

9.3.2. Wyjątki bazodanowe po stronie serwerowej

Drugi sposób obsługi sytuacji problematycznych związanych z bazą danych opiera się na walidacji rezultatów zapytań bazodanowych. Dla przykładu: zapytanie do bazy danych zwraca pusty rezultat. Jednak oczekiwaną wartością wspomnianej operacji jest dokładnie jeden wynik. W konsekwencji moduł bazodanowy realizuje walidację sprawdzającą czy rozmiar wyniku jest dokładnie równy jeden, co powoduje rzucenie wyjątku informującego o problemie.
Każdy z przytoczonych wyżej sposobów zwraca do części serwerowej informację o błędzie wykonania. W celu zapewnienia pełnej niezależności wpomnianych modułów problemy przetwarzania danych są przechwytywane w części bazodanowej i tłumaczone na odpowiadające im wyjątki zdefiniowany w module DatabaseInterfaces.

9.4. Wyjątki serwerowe

Wszystkie wyjątki rzucane po stronie serwera są wyjątkami protokołu ICE i zostały zdefiniowane w plikach Slice (każdy zawiera kompletną wiadomość dla użytkownika końcowego) i dziedziczą po TudasServerException. Obecnie wykorzystywane są następujące rodzaje wyjątków:

- *ArgumentFormatException* - związany z błędem walidacji argumentów wywołania którejś z zdalnych metod, np. pusta kolekcja, ujemna wartość znacznika czasowego.
- *TudasDatabaseException* - reprezentuje ogół wyjątków bazodanowych (więcej szczegółów w sekcji *Wyjątki bazodanowe* tego rozdziału)

27 SqlException - rodzaj wyjątku rzucany przez framework spring jdbc w przypadku gdy został zwrócony kod błędy z bazy danych

- *SecurityException* - bazowa klasa wyjątków związanych z bezpieczeństwem, tj. uwierzytelnianiem i autoryzacją zapytań. Pochodne wyjątki:
    - *TicketNotValid* - rzucany, gdy ticket nie mógł zostać poprawnie odszyfrowany lub gdy nie jest ważny. Więcej szczegółów w rodziale 7.
    - *CredentialsAuthenticationFailureException* - pojawia się, gdy nastąpił błąd uwierzytelniania danych dostępowych użytkownika w systemie CERN SSO
    - *PermissionsRequiredException* - związany jest z modułem autoryzacji i oznacza, że użytkownik próbujący wykonać jakąś operację nie ma do tego wystarczających uprawnień

9.5. Wyjątki klienckie

W przypadku modułu klienta występują zarówno wyjątki zdefiniowane w Slice, jak i klasyczne wyjątki zapisane w danym języku programowania. Sprowadza się to do możliwości wystąpienia trzech typów wyjątków:

- TudasClientException -zdefiniowany w Slice, nie jest wykorzystywany w warstwie komunikacji, ale ze względu na zgodność w implementacji konwertera wyjątków jest wygodny w użyciu w sytuacji błędów związanych z tworzeniem obiektu proxy lub nawiązywaniem połączenia z serwerem.
- *TudasException* - wyjątek zdefiniowany w kodzie systemu, przeznaczony jest do interakcji z użytkownikiem na ostatnim etapie obsługi sytuacji wyjątkowych. Zawiera komplet informacji o naturze wyjątku, jego przyczynie i ewentualnym sugerowanym rozwiązaniu problemu. Posiada jeden wyjątek pochodny:
    - *WorkspaceException* - dotyczy sytuacji związanych z konfiguracją przestrzeni roboczej i zwykle oznacza błędy, pojawiające się podczas interakcji z systemem plików użytkownika (błąd otwarcia/zapisu do pliku itp)

10. Podsumowanie

W poprzednich rozdziałach zostały opisane wszystkie najważniejsze aspekty systemu TUDAS. W ramach podsumowania warto przytoczyć perspektywy dalszego rozwoju projektu oraz zweryfikować przyjęte wymagania i cele.

10.1. Dalszy rozwój

Zrealizowany system spełnia wszystkie pierwotnie postawione przed nim wymagania funkcjonalne. Jednakże istnieją możliwości jego dalszego rozwoju. Pod koniec prac klienci zasugerowali dodatkowe pomysły dotyczące rozwoju funkcjonalności: dodanie kolejnych interfejsów oraz zrefaktorowanie bazy, aby umożliwić przechowywanie dodatkowych danych.

Zastosowanie dodatkowych technologii i koncepcji pozwoli na potencjalne zwiększenie wydajności systemu. Klasteryzacja serwera przy pomocy narzędzia IceGrid[28] pozwoli nie tylko zwiększyć przepustowość systemu poprzez równoważenie obciążania, ale także jego niezawodność poprzez replikację. W przypadku klasteryzacji, konieczne będzie odseparowanie pamięci podręcznej od części serwerowej, aby każdy węzeł miał dostęp do danych wcześniej zapamiętanych.

10.2. Weryfikacja wymagań

Cechą chrarakterystyczną projektu była zdecydowana przewaga liczby wymagań niefunkcjonalnych nad funkcjonalnymi. System musiał zapewnić takie aspekty jak niezawodność, częściowa niezależność od platformy, rozszerzalność, prostota obsługi i efektywność działania. Wszystkie wspomniane wymagania zostały sprawdzone, przetestowane i opisane, jednakże dopiero po pełnym, planowanym wdrożeniu projektu będzie można stwierdzić, że zrealizowany system w tym względzie odniósł pełny sukces. Po zrealizowaniu wszystkich wymagań funkcjonalnych została wdrożona testowa wersja systemu do użytku klientów. Naukowcy zaakceptowali wszystkie interfejsy zaproponowane przez zespół projektowy. Przeprowadzone rozmowy z klientami wykazały, że zaoferowane rozwiązania znacząco przyczynią się do poprawy komfortu i efektywności pracy. Przeprowadzone testy integracji systemów TUDAS i DBPop wskazały, że system realizuje stawiane przed nim oczekiwania. Integracja z systemem DBEL nie doszła do skutku, ponieważ zmiany przeprowadzone na bazie danych spowodowały, że wspomniany system nie realizuje oczekiwanych funkcjonalności. Natomiast przykładowe aplikacje klienckie wskazały na poprawną realizację funkcjonalności przez system TUDAS.

28 ICE Grid - narzędzie dostarczające funkcjonalności zorientowanego obiektowo równoważenia obciążenia, obsługi błędów, wykrywania obiektów oraz rejestru usług.