



Politechnika Gdańska
Wydział Elektroniki,
Telekomunikacji i Informatyki



Katedra: Architektury Systemów Komputerowych

Imię i nazwisko dyplomanta: Marcel Schally - Kacprzak

Nr albumu: 113783

Forma i poziom studiów: Stacjonarne jednolite studia magisterskie

Kierunek studiów: Informatyka

Praca dyplomowa magisterska

Temat pracy:
Graficzny edytor aplikacji z odwzorowaniem na CPU i GPU

Kierujący pracą:
dr inż. Paweł Czarnul

Zakres pracy:
Celem pracy jest zaprojektowanie i zaimplementowanie graficznego narzędzia, pozwalającego w łatwy i intuicyjny sposób na tworzenie scenariuszy, składających się z zadań obliczeniowych oraz relacji czasowych między nimi

Gdańsk, 2012

Spis treści

1	Wstęp	7
1.1	Analiza problemu	7
1.2	Cel pracy	9
1.3	Przyjęte rozwiązanie	9
2	Analiza istniejących rozwiązań	11
2.1	Krótkie streszczenie rozdziału	11
2.2	Klasyfikacja podejść i rozwiązań	11
2.2.1	Podział ze względu na stopień wspomagania wykonywania obliczeń	11
2.3	Wymagana wiedza i narzędzia	12
2.3.1	Znajomość projektowania aplikacji graficznych	12
2.3.2	Znajomość technik komunikacji sieciowej	13
2.3.3	Znajomość charakterystyki systemów gridowych	13
2.3.4	Znajomość charakterystyk języków opisujących scenariusze (ang. <i>workflow</i>)	13
2.4	Przykłady rozwiązań	13
2.4.1	BeesyCluster	13
2.4.2	Cloudera	14
2.4.3	Pegasus	15
2.4.4	GridAnt	16
2.4.5	Triana	17
2.4.6	ICENI	17
2.4.7	UNICORE	18
2.4.8	Taverna	18

2.4.9	Kepler	19
2.4.10	CAT	19
2.4.11	JOpera	20
2.4.12	Activiti	21
2.4.13	Bonitasoft	22
2.4.14	Joget Workflow	22
2.4.15	ProcessMaker	23
3	Kontekst systemu „Kernel Hive”	25
3.1	Krótkie streszczenie rozdziału	25
3.2	Opis systemu „Kernel Hive”	25
3.3	Architektura systemu „Kernel Hive”	26
3.4	Komponenty systemu „Kernel Hive”	27
4	Specyfikacja wymagań systemowych	29
4.1	Krótkie streszczenie rozdziału	29
4.2	Specyfikacja wymagań systemowych	29
4.2.1	Udziałowcy	29
4.2.2	Cele systemu	30
4.2.3	Otoczenie systemu	31
4.2.4	Przewidywane komponenty systemu	32
4.2.5	Wymagania	34
4.2.6	Sytuacje wyjątkowe	36
4.2.7	Dodatkowe wymagania	36
4.2.8	Kryteria akceptacyjne	37
4.2.9	Słownik	38
4.3	Główne przypadki użycia	38
4.3.1	Diagram głównych przypadków użycia	38
4.3.2	Opis głównych przypadków użycia	39
5	Projekt systemu	43
5.1	Krótkie streszczenie rozdziału	43

Spis treści	5
5.2 Analiza wymagań architektonicznych	43
5.2.1 Wymagania architektoniczne przypadków użycia	43
5.2.2 Wybór przypadków użycia sterujących architekturą systemu	45
5.3 Projekt architektury systemu	45
5.3.1 Koncepcja architektury systemu	45
5.3.2 Schemat architektury systemu	46
5.3.3 Komponenty sprzętowe	46
5.3.4 Podsystemy	46
5.3.5 Warstwy architektoniczne	49
5.3.6 Komponenty programowe	49
5.3.7 Sposób realizacji wymagań jakościowych	50
6 Problemy implementacyjne	53
6.1 Krótkie streszczenie rozdziału	53
6.2 Metodologia projektowania i implementacji	53
6.2.1 Metodologia projektowania systemu	53
6.2.2 Metodologia implementacji	54
6.3 Środowiska i narzędzia wspomagające	54
6.4 Główne problemy implementacyjne	56
6.5 Przykłady rozwiązań szczegółowych	57
6.5.1 Sposób serializacji i deserializacji scenariusza obliczeniowego	57
6.5.2 Komunikacja z Silnikiem Wykonawczym	66
6.5.3 Komunikacja z Repozytorium Szablonów Obliczeniowych	71
6.5.4 Edycja kodu źródłowego metod obliczeniowych	72
6.5.5 Sposób tworzenia scenariuszy obliczeniowych	73
7 Realizacja i ocena systemu	77
7.1 Krótkie streszczenie rozdziału	77
7.2 Docelowa architektura i opis komponentów	77
7.2.1 Docelowa architektura aplikacji	77
7.2.2 Opis podsystemów	78

7.2.3	Opis warstw architektonicznych	82
7.2.4	Opis komponentów programowych	83
7.2.5	Główne przykłady interakcji aplikacji z systemem „Kernel Hive”	83
7.3	Metodologia systemowa	90
7.4	Wyniki oceny	90
7.5	Możliwości rozbudowy	90
8	Uwagi końcowe	93
8.1	Warte podkreślenia osiągnięcia natury informacyjnej	93
8.2	Perspektywy wykorzystania systemu oraz możliwości wdrożeniowe	93
9	Bibliografia	95
10	Załączniki	97

Rozdział 1

Wstęp

1.1 Analiza problemu

W latach 80. XX wieku komputery powszechnie wkroczyły do dziedzin naukowych. I głównym celem było, i nadal jest, służyć jako maszyny modelujące badane zjawiska - fizyczne, chemiczne, czy biologiczne. Z upływem lat komputery stawały się coraz wydajniejsze, jednak eksperymenty naukowe stawiały coraz wyższe poprzeczki przed możliwościami obliczeniowymi maszyn. Pomysł połączenia wielu komputerów za pomocą sieci w system gridowy pojawił się już w latach 90. XX wieku, wprowadzony przez Iana Fostera, Carla Kesselmana i Steve'a Tuecke'a.

Mimo wprowadzenia wielu narzędzi i bibliotek ułatwiających wytwarzanie aplikacji na te systemy, stopień ich skomplikowania wymagał głębokiej wiedzy z zakresu komunikacji sieciowej i specyfiki środowisk przetwarzania rozproszonego. Wymagane było wymyślenie innego podejścia do projektowania takich aplikacji, takie, które nie wymagałyby od użytkownika wiedzy na temat niskopoziomowej specyfiki danego środowiska i pozwalałyby mu skupić się na osiągnięciu celów biznesowych.

Koncepcja scenariusza w kontekście domeny biznesowej została przedstawiona w 1996 roku przez WMC (ang. *Workflow Management Coalition*) jako:

(ang.) *"The automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules."*

Scenariusze pojawiły się jako odpowiedź na problem zbytniego skomplikowania procesów biznesowych lub eksperymentów naukowych, dostarczając sposób deklaratywnego opisu *co* dana aplikacja ma osiągnąć, a nie *jak* będzie wykonywana.

Różne rodzaje zadań, które można wykonać w ramach scenariusza, są odpowiedzialne tylko za mały fragment funkcjonalności danego scenariusza, dlatego wymagane jest łączenie ich w większe struktury, tak by w efekcie otrzymać scenariusz wykonujący jaką pożyteczną funkcję.

Definiowanie, wykonanie i monitorowanie scenariusza zarządzane jest za pomocą Systemu Zarządzającego Scenariuszem (ang. *"Workflow Management System"*), systemem zapewniającym środowisko, w którym można powyższe operacje wykonywać. Ta dodatkowa warstwa, zapewniająca nam zarządzanie przechowywaniem i przesyłaniem danych, bezpieczeństwem, monitorowaniem, pozwala nam na wykonywanie scenariuszy w systemach gridowych, nie zajmując się szczegółami komunikacyjnymi pomiędzy zadaniami, skupiając się na logice biznesowej aplikacji i zaprojektowaniu jej architektury.

Dzięki wykorzystaniu warstwy zarządzającej scenariuszami wykonywanymi w systemach gridowych, ich tworzenie sprowadza się do definiowania elementów wykonawczych scenariusza (zadań) i określania zależności między nimi. Dzięki takiemu podejściu użytkownicy nieposiadający specjalistycznej wiedzy z zakresu systemów rozproszonych (na przykład naukowcy) mogą w pełni wykorzystać możliwości systemów gridowych.

Wprowadzenie paradygmatu scenariuszy do tworzenia aplikacji na systemy gridowe posiada szereg zalet:

- Możliwość budowy dynamicznych aplikacji które łączą rozproszone zasoby
- Wykorzystanie zasobów należących do danej domeny by zwiększyć wydajność lub obniżyć koszty
- Integracja wielu zespołów włączonych w zarządzanie i rozwój części składowych scenariusza, wspierając tym samym współpracę między różnymi jednostkami administracyjnymi
- Wykonanie scenariusza rozciąga się na kilka domen administracyjnych by uzyskać określone możliwości przetwarzania

1.2 Cel pracy

Celem pracy jest zaprojektowanie i zaimplementowanie graficznego narzędzia, pozwalającego w łatwy i intuicyjny sposób na tworzenie scenariuszy, składających się z zadań obliczeniowych oraz relacji czasowych między nimi. W ten sposób użytkownik będzie w stanie graficznie zaprezentować scenariusz jako acykliczny graf skierowany, w którym wierzchołki grafu są zadaniami, a krawędzie - relacjami czasowymi między nimi. Tak utworzony graf będzie można następnie przesłać do wykonania do Silnika Wykonawczego systemu, który automatycznie wykona potrzebne obliczenia, korzystając z dostępnych zasobów i mając na uwadze zależności między poszczególnymi zadaniami.

Działanie narzędzia będzie polegało na umożliwieniu wykonania następujących czynności użytkownikowi:

1. Utworzenie nowego projektu
2. Dodanie węzłów (zadań) do grafu (scenariusza) i określenie zależności między nimi.
3. Zdefiniowanie ciał metod zadań obliczeniowych
4. Wysłanie utworzonego scenariusza oraz danych wejściowych do systemu wykonawczego
5. Odebranie wyników zakończonego wykonania scenariusza

1.3 Przyjęte rozwiązanie

Zdecydowano się na zaimplementowanie narzędzia w postaci aplikacji desktopowej. Ponieważ wieloplatformowość aplikacji jest istotnym kryterium użyteczności, zdecydowano się na wykorzystanie języka Java oraz biblioteki graficznej Swing. Dzięki temu aplikacja nie będzie wymagać żadnego dodatkowego oprogramowania ponad to powszechnie spotykane na komputerach klasy PC.

Rozdział 2

Analiza istniejących rozwiązań

2.1 Krótkie streszczenie rozdziału

Na rynku istnieje wiele narzędzi, zarówno bezpłatnych jak i płatnych, wspomagających wykonywanie rozproszonych obliczeń w systemach gridowych. Znacząca większość rozwijana jest przez instytucje naukowe, obecne są również jednak rozwiązania komercyjne jak i systemy rozwijane przez społeczności Open Source.

2.2 Klasyfikacja podejść i rozwiązań

Istnieje wiele kategorii podziału oprogramowania odpowiadającego za wspomaganie wykonywania rozproszonych obliczeń w systemach gridowych. W pracy zdecydowano się na klasyfikację rozwiązań na podstawie następujących kryteriów:

1. Podział ze względu na stopień wspomagania wykonywania obliczeń

2.2.1 Podział ze względu na stopień wspomagania wykonywania obliczeń

Wśród rozpatrywanego oprogramowania można wyróżnić 3 podstawowe grupy:

1. Kompletnie, zintegrowane środowiska wykonawcze
2. Narzędzia wspomagające pracę z systemami rozproszonymi
3. Zbiory komponentów programowych

Kompletne, zintegrowane środowiska wykonawcze

Cześć z narzędzi wspomagających pracę z systemami rozproszonymi jest dystrybuowana w postaci kompletnych środowisk wykonawczych, zawierających w sobie nie tylko komponenty odpowiedzialne za wspomaganie wykonywania obliczeń rozproszonych, ale również właściwy system wykonujący te obliczenia. Przykładem takich środowisk mogą być np. Triana, Pegasus, UNICORE, Taverna, Kepler.

Narzędzia wspomagające pracę z systemami rozproszonymi

Na rynku obecne są również narzędzia oferujące samą funkcję wspomaganie pracy z systemami rozproszonymi. Różnią się możliwościami oraz złożonością, lecz posiadają wspólne cechy - na przykład umożliwiają wgląd i kontrolę nad stanem systemu wykonawczego oraz jego systemem zabezpieczeń, lub też wspomagają użytkownika podczas tworzenia scenariuszy obliczeniowych. Przykładem takich rozwiązań jest m. in. BeesyCluster, Cloudera, ICENI.

Zbiory komponentów programowych

Osobną grupą są narzędzia będące tak naprawdę zbiorem komponentów programowych, których efektywne wykorzystanie wymaga od użytkownika wiedzy z dziedziny informatyki. Reprezentują sobą największą elastyczność, jednak jest to okupione wyższym kosztem uzyskania użyteczności. Przykładem tego typu narzędzi jest GridAnt, będący frameworkiem do prototypowania aplikacji gridowych, bazując na narzędziu Ant.

2.3 Wymagana wiedza i narzędzia

W tej sekcji skupiono się na wiedzy i narzędziach potrzebnych do zaprojektowania narzędzia wspomagającego wykonywanie rozproszonych obliczeń, a nie potrzebne do zaprojektowania samych systemów rozproszonych.

2.3.1 Znajomość projektowania aplikacji graficznych

Podczas tworzenia narzędzia wspomagającego wykonywanie rozproszonych obliczeń konieczna jest znajomość jednej z bibliotek graficznych - prawidłowo wykonany interfejs użytkownika, prosty i intuicyjny, jest jedną z ważniejszych cech produktu.

2.3.2 Znajomość technik komunikacji sieciowej

Narzędzie wspomagające wykonywanie rozproszonych obliczeń deleguje faktyczne ich wykonanie do systemu wykonawczego (np. do systemu gridowego). Wiedza na temat komunikacji sieciowej, wysyłaniu i odbieraniu danych i komunikatów jest tu nieodzowna.

2.3.3 Znajomość charakterystyki systemów gridowych

Wiedza z zakresu systemów gridowych i ich sposobu działania jest nieodzowna do zaprojektowania narzędzia wspomagającego wykonywanie na nich obliczeń.

2.3.4 Znajomość charakterystyk języków opisujących scenariusze (ang. *workflow*)

W celu prawidłowego zamodelowania scenariusza obliczeniowego a następnie wykonania go na systemie gridowym niezbędna jest znajomość języków opisu scenariuszy, ich specyfiki, możliwości i ograniczeń.

2.4 Przykłady rozwiązań

2.4.1 BeesyCluster

BeesyCluster jest portalem, będącym punktem dostępowym do sieci klastrów/komputerów klasy PC. Dzięki systemowi wirtualnych kont, użytkownik uzyskuje dostęp do jego wszystkich kont fizycznych na maszynach będących częścią systemu. BeesyCluster posiada interfejs WWW jak i WebServices, co stanowi świetne rozwiązanie obejmujące użytkowników komputerów stacjonarnych i urządzeń przenośnych.

Zarejestrowanym użytkownikom, BeesyCluster udostępnia następujące funkcje (poprzez WebServices - w ograniczonym zakresie):

- Zdalne zarządzanie wszystkimi kontami fizycznymi w systemie, w szczególności kopiowanie, tworzenie, edycję, podgląd, (de)kompresję plików, interaktywne i kolejkowe wykonywanie zadań
- Publikacja usług przez użytkowników i udostępnienie ich innym
- Uruchamianie usług innych użytkowników

- Zarządzanie systemem uprawnień usług
- Środowisko pracy zespołowej - wysyłanie wiadomości i komunikatów innym użytkownikom. rysowanie na współdzielonej tablicy, itp.
- łączenie dostępnych usług w scenariusze, optymalizacja doboru usług do scenariusza, wykonanie scenariusza

System BeesyCluster został stworzony na Wydziale Elektroniki, Telekomunikacji i Informatyki Politechniki Gdańskiej, na Katedrze Architektury Systemów Komputerowych.

2.4.2 Cloudera

Cloudera Apache Hadoop Manager jest aplikacją zarządzającą systemem Apache Hadoop¹. Po-
przez dostarczenie użytkownikowi wglądu i kontroli nad każdą częścią składową klastra Hadoop,
Cloudera pozwala na polepszenie wydajności klastra, zwiększa QoS (ang. *Quality of Service*) i
redukuje koszty administracyjne.

Cloudera Manager pozwala na łatwe wdrażanie aplikacji i centralną obsługę całego stosu Ha-
doop. Automatyzuje proces instalacji, redukując czas wdrożenia. Cloudera udostępnia podgląd
wszystkich usług i hostów działających w klastrze w czasie rzeczywistym. Jest jednocześnie central-
nym punktem wprowadzania zmian w konfiguracji, propagowanej następnie na wszystkie maszyny
klastra. Cloudera zawiera w sobie cały szereg narzędzi diagnostycznych i raportowych wspomag-
ających użytkownika w optymalizacji wydajności i wykorzystania klastra.

Cloudera Manager udostępnia kompleksowe zarządzanie klastrem Apache Hadoop:

- Instaluje cały stos Hadoop za pomocą "wizarda"
- Daje kompletną kontrolę nad klastrem Hadoop za pomocą pojedynczego interfejsu
- Umożliwia ustawienie ról systemowych i konfigurację usług w klastrze
- Umożliwia "łagodny"(ang. *graceful*) start, zatrzymanie i restart usług.
- Prezentuje informacje dotyczące hostów w klastrze, łącznie z ich statusem, pamięcią fizyczną i wirtualną, rolami.

Zarządzanie odbywa się praktycznie w całości za pomocą Cloudera Manager Admin Console -
interfejsu udostępnianego przez protokół HTTP, podłączonego do Cloudera Manager Server.

¹<http://hadoop.apache.org/>

2.4.3 Pegasus

Projekt Pegasus² zawiera w sobie zbiór technologii, które umożliwiają uruchamianie scenariuszy (ang. *workflow*) w wielu różnych środowiskach - desktopach, klastrach, systemach gridowych oraz chmurach. Scenariusze naukowe dają użytkownikowi możliwość łatwego opisania wielokrokowych obliczeń, przykładowo pobrania danych z bazy danych, przeformatowania ich, a następnie przeprowadzenia stosownej analizy. Kiedy aplikacja zostaje opisana jako scenariusz, Usługa Zarządzania Scenariuszami Pegasus (ang. *Pegasus Workflow Management Service*) może rzutować ją na dostępne zasoby obliczeniowe i wykonać jej kroki w zadanym porządku. Pegasus swobodnie radzi sobie z scenariuszami liczącymi kilka milionów zadań obliczeniowych.

Pegasus jest używany w wielu dziedzinach naukowych, łącznie z astronomią, bioinformatyką, sejsmologią, fizyce fal grawitacyjnych, oceanografią, limnologią, i innymi. Jeśli podczas wykonywania obliczeń wystąpi błąd, Pegasus stara się (jeśli to możliwe) wykonać zadanie jeszcze raz, wykonać jeszcze raz cały scenariusz, lub, jeśli to zawiedzie, utworzyć scenariusz ratunkowy, zawierający opis jedynie tej części obliczeń, która nie została wykonana. Wykonuje również okresowe zwalnianie używanych zasobów pamięciowych, by scenariusze przetwarzające wielkie wolumeny danych miały zawsze wystarczająco miejsca do wykonania obliczeń. Pegasus śledzi i zapamiętuje wykonywane zadania, łącznie z lokalizacją danych użytych i utworzonych oraz użytym przez nie oprogramowaniem.

Pegasus łączy naukową dziedzinę i środowisko wykonawcze poprzez automatyczne mapowanie scenariuszy o wysokim poziomie abstrakcji na rozproszone zasoby. Wykrywa i lokalizuje potrzebne dane wejściowe i zasoby obliczeniowe potrzebne do wykonania zadania. Pegasus umożliwia naukowcom tworzenie abstrakcyjnych scenariuszy, uwalniając od martwienia się o parametry środowiska wykonawczego czy niskopoziomą specyfikację warstwy pośredniej (ang. *middleware*) - na przykład Globus, Condor, Amazon EC2). Pegasus spina również dostępną infrastrukturę sieciową, efektywnie koordynując rozproszone zasoby.

Pegasus składa się z trzech komponentów:

- Mapper (Pegasus Mapper):

Generuje scenariusz wykonawczy bazujący na abstrakcyjnym scenariuszu dostarczonym przez użytkownika. Wyszukuje potrzebne dane, oprogramowanie, i zasoby obliczeniowe do wykonania zadanego scenariusza. Mapper może również zrestrukturyzować scenariusz w celu optymalizacji wydajności.

²<http://pegasus.isi.edu/>

- Silnik Wykonawczy (DAGMan³):
Wykonuje zadania zdefiniowane przez scenariusz w kolejności związanej z ich zależnościami. DAGMan wykorzystuje zasoby (obliczeniowe, pamięciowe, sieciowe) zdefiniowane w scenariuszu wykonawczym by wykonać potrzebne operacje.
- Manager zadań (Condor Schedd⁴):
Zarządza pojedynczymi zadaniami scenariusza; nadzoruje ich wykonanie na lokalnych i zdalnych zasobach.

Projekt Pegasus jest rozwijany przez Wydział Informatyki Uniwersytetu Madison w Wisconsin i USC Information Sciences Institute.

2.4.4 GridAnt

GridAnt udostępnia prosty, lecz rozszerzalny framework do prototypowania aplikacji gridowych na podstawie specyfikacji w języku XML (ang. *eXtensible Markup Language*). Podstawowe zadania, takie jak uwierzytelnianie, transfer plików czy wykonanie zadania są predefiniowane. Jak sama nazwa wskazuje, implementacja GridAnt bazuje na narzędziu Ant⁵. Dzięki temu umożliwia użycie cech Ant'a, jak na przykład XML'owe specyfikacje zadań, specyfikację przepływu przez warunkowe, sekwencyjne i równoległe konstrukcje frameworka oraz dostępność silnika przetwarzającego scenariusze. Ponieważ GridAnt dostarcza abstrakcyjną definicję zadań elementarnych wykonywanych w gridach, umożliwia tym samym uruchomienie tej samej specyfikacji aplikacji na różnych środowiskach gridowych - przykładowo uruchomienie takiej aplikacji na Globus Toolkit⁶ 2 i 3.

GridAnt jest świetnym narzędziem, nie tylko do mapowania skomplikowanych scenariuszy ale również jako prosty klient do testowania działania różnych usług w systemach typu grid. Jak twierdzą twórcy, nie był pisany dla zastąpienia bardziej skomplikowanych silników, jak na przykład BPEL4WS⁷, XLANG⁸, WSFL⁹. Filozofia stojąca za GridAnt zakłada wykorzystanie silnika Apache Ant i rozwinięcie słownika scenariuszy dla systemów gridowych.

³<http://research.cs.wisc.edu/condor/dagman/>

⁴<http://research.cs.wisc.edu/condor/>

⁵<http://ant.apache.org/>

⁶<http://www.globus.org/toolkit/>

⁷<http://www.ibm.com/developerworks/library/specification/ws-bpel/>

⁸<http://www.ebpm1.org/xlang.htm>

⁹<http://xml.coverpages.org/wsfl.html>

2.4.5 Triana

Triana to wolne środowisko do planowania i wykonywania scenariuszy rozwijane przez Cardiff University, które łączy w sobie graficzny interfejs i narzędzia do analizy danych. Jest używane przez naukowców do szeregu różnych zadań, jak na przykład przetwarzanie sygnału, tekstu, obrazu. Triana zawiera w sobie sporych rozmiarów bibliotekę narzędzi analitycznych i pozwala na łatwą integrację z modułami napisanymi przez użytkownika.

Triana udostępnia graficzne środowisko pozwalające na tworzenie i używanie programów, mając minimalną wiedzę i umiejętności z zakresu programowania. Budowa aplikacji polega na wykorzystaniu zbioru predefiniowanych bloków programowych, przeciąganiu ich na przestrzeń roboczą i łączeniu z innymi przy pomocy myszy komputerowej. Triana została napisana w języku Java, co pozwala na uruchomienie jej na prawie każdym komputerze.

Triana jest rozwijana przez zespół naukowy z Cardiff University w Wielkiej Brytanii. Prace prowadzone są w ramach eksperymentu GEO600 badającego fale grawitacyjne. Badania prowadzone są przy współpracy z naukowcami z Niemiec, Anglii i innych krajów. GEO600 ma generować kilka terabajtów danych na rok, a Triana ma pomóc naukowcom w analizie tych danych w prosty i uniwersalny sposób.

2.4.6 ICENI

ICENI (ang. *Imperial College e-Science Networked Infrastructure*) jest usługowo zorientowanym oprogramowaniem udostępniającym komponentowy model programistyczny, który programista może wykorzystać do zaprojektowania aplikacji na systemy gridowe; jest również infrastrukturą wykonawczą, która określa zasoby klastra (obliczeniowe, pamięciowe i programowe) jako usługi określone warunkami, definiującymi kto i jak może skorzystać z zasobów.

Celem twórców jest zapewnienie dla naukowców infrastruktury o wysokim poziomie abstrakcji, pozwalającej użytkownikom na skonstruowanie i określenie ich własnych aplikacji za pomocą graficznego narzędzia kompozycyjnego, zintegrowanego z rozproszonymi repozytoriami komponentów programowych. Repozytoria komponentów będą pozwalały programistom na dzielenie się własną pracą z resztą społeczności i na łatwe użycie cudzych komponentów w tworzonej aplikacji.

ICENI został napisany w języku Java i Jini (obecnie Apache River¹⁰).

¹⁰river.apache.org/

2.4.7 UNICORE

UNICORE (ang. *Uniform Interface to Computing Resources*) to, jak twierdzą twórcy, gotowy do uruchomienia system gridowy składający się z oprogramowania po stronie serwera i klienta (dostępny "gruby klient", jak również interfejs linii komend). System wydany jest na licencji BSD oraz opiera się na powszechnie używanych standardach, takich jak *Open Grid Services Architecture*¹¹ (OGSA) i *Web Services Resource Framework*¹² (WS-RF 1.2). Wykorzystuje architekturę SOA, co pozwala na łatwe zastąpienie jednych komponentów innymi.

Wiele pracy poświęcono bezpieczeństwu - system używa certyfikatów X.509 jako podstawę uwierzytelnienia i autoryzacji, dodatkowo obsługuje certyfikaty serwerów proxy i użycie wirtualnych organizacji (VO).

2.4.8 Taverna

Taverna jest wolnym systemem zarządzania scenariuszami - zbiorem narzędzi do zaprojektowania i wykonania naukowych scenariuszy. Taverna jest napisana w języku Java, i zawiera w sobie następujące komponenty:

- Taverna Engine, silnik używany do wykonywania scenariuszy
- Taverna Workbench, desktopowa aplikacja kliencka
- Taverna Server, serwer pozwalający na zdalne wykonywanie scenariuszy
- Taverna Command Line Tool, narzędzie umożliwiające wykonywanie prostych operacji z linii komend

Taverna pozwala na automatyzację procesu obliczeń danych na potrzeby eksperymentów poprzez udostępnienie różnych (lokalnych, jak i zdalnych) usług z szeregu naukowych dziedzin - biologii, chemii, medycyny, inżynierii dźwięku, meteorologii, lub socjologii. W rezultacie, umożliwia naukowcom z niewielką wiedzą z dziedzin informatycznych i ograniczonymi zasobami obliczeniowymi konstruowanie złożonych scenariuszy a następnie na ich uruchamianie ze zwykłej, biurowej stacji roboczej.

¹¹<http://www.globus.org/ogsa/>

¹²<http://www.globus.org/wsrf/>

2.4.9 Kepler

Kepler jest projektem skierowanym do naukowców, zawierającym w sobie graficzny interfejs użytkownika do tworzenia scenariuszy i silnik wykonawczy uruchamiający scenariusze. Użytkownicy mogą tworzyć, wykonywać i dzielić się scenariuszami i ich komponentami z innymi. Kepler może działać na wielu formatach danych, lokalnie, jak i zdalnie.

Tworzenie scenariuszy polega na mechanizmie *drag-and-drop* - użytkownik wybiera interesujące go komponenty, łączy je ze sobą i ze źródłami danych tworząc spójny scenariusz. Kepler wspomaga ponowne wykorzystanie elementów składowych scenariuszy i dzielenie się nimi między użytkownikami. Scenariusze tworzone za pomocą projektu mogą być zagnieżdżane, co pozwala na tworzenie złożonych przepływów z prostszych komponentów oraz na tworzenie podstawowych scenariuszy, które mogą następnie być zapisane i ponownie użyte w większych strukturach.

Kepler jest oparty na systemie Ptolemy II¹³, dojrzałej platformie wspierającej wiele modeli obliczeniowych pasujących do różnych typów obliczeń i analiz (przetwarzanie danych z sensorów lub, na przykład, całkowanie równań różniczkowych).

Projekt Kepler jest rozwijany i utrzymywany przez zespół, w skład którego wchodzi członkowie kilku instytucji, w tym Davis University, Santa Barbara University, Sand Diego University. Główną odpowiedzialność za projekt ponosi Zespół Przywódczy (ang. *Leadership Team*), którego zadaniem jest zapewnienie długoterminowej technicznej i finansowej stabilności.

Kepler jest aplikacją opartą na języku Java.

2.4.10 CAT

CAT¹⁴ (ang. *Composition Analysis Tool*) jest opartym na ontologii interfejsem do kompozycji scenariuszy. Elementami takich scenariuszy są tutaj usługi sieciowe oferowane przez różnych dostawców, które można łączyć w dowolny sposób tak, by osiągnąć zamierzony cel biznesowy.

CAT jest adresowany do zastosowania w rozwijającej się dziedzinie aplikacji badawczych, które są wynikiem złączenia wielu pojedynczych modeli naukowych. Niewspomagane utworzenie scenariusza takiej aplikacji nakłada spore wymagania na użytkownika, jak na przykład wiedza, w jaki sposób duża ilość skomplikowanych zadań może być łączona ze sobą. Z drugiej strony, w

¹³<http://ptolemy.eecs.berkeley.edu/ptolemyII/>

¹⁴<http://www.isi.edu/ikcap/cat/>

pełni automatyczny planista może utworzyć wiele tak samo poprawnych scenariuszy. Skrócenie wyszukiwania optymalnego scenariusza może być zrealizowane poprzez krokową interakcję z użytkownikiem, który może określać swoje preferencje.

Innowacyjność tego rozwiązania polega na inteligentnym silniku analizującym utworzony scenariusz, oferującym sugestie naprawy błędów kompilacji i rozplanowania zależności między usługami. Silnik utrzymuje również bazę wiedzy na temat wszystkich dostępnych komponentów, o ich właściwościach i typach danych wejściowych i wyjściowych. Korzysta z niej podczas sprawdzania, czy utworzony scenariusz jest zgodny z formalnie zdefiniowanym zbiorem oczekiwanych właściwości.

Narzędzie CAT jest rozwijane, by wspomóc tworzenie skomplikowanych naukowych scenariuszy w projekcie SCEC/IT¹⁵ w ramach badań nad trzęsieniami ziemi. Jednostką odpowiedzialną za finansowanie rozwoju aplikacji jest National Science Foundation (NSF).

2.4.11 JOpera

JOpera¹⁶ jest narzędziem do kompozycji scenariuszy opartym na środowisku Eclipse. Elementami scenariuszy są tutaj (głównie, lecz nie tylko) usługi sieciowe. JOpera oferuje wizualny język opisu scenariusza oraz autonomiczną platformę wykonawczą do ich uruchamiania. Narzędzie wspomaga użytkownika w kilku kwestiach:

- Komponowanie usług metodą Bottom-up i top-down
- Heterogeniczność wykorzystywanych usług - można wywołać usługi sieciowe SOAP, REST, servlety Java, skrypty JavaScript, usługi gridowe i inne.
- Integracja modelowania procesu i wykonania

JOpera umożliwia:

- Wizualną definicję tworzonego procesu - wspomaga deklarowanie przepływu kontroli, ale i modeluje przepływ danych
- Efektywne wykonanie scenariusza - jego wizualny model zostaje skompilowany do kodu bajtowego Javy
- Monitorowanie wykonywania scenariusza - narzędzie umożliwia interakcyjne sterowanie wykonaniem i jego debugowanie

¹⁵<http://www.isi.edu/ikcap/scec-it/>

¹⁶<http://www.jopera.org/>

- Rekursywną kompozycję usług - automatycznie publikuje nowo utworzony scenariusz jako usługę sieciową w standardzie SOAP/WSDL i/lub REST

W ramach projektu JOpera badane są również następujące problemy:

- Zarządzanie scenariuszem w chmurze
- Komponowanie usług REST
- Architektury zorientowane strumieniowo (ang. *Streaming Oriented Architectures*) - JOpera przetwarza zamodelowane rurociągi (ang. *pipelines*) i mapuje je na generyczne strumienie danych i usługi strumieniowe
- Zarządzanie zmianą interfejsów usług - JOpera określa, które procesy będą dotknięte zmianą definicji usług
- Śledzenie rodowodu danych - metadane pochodzenia danych są automatycznie zbierane przez silnik wykonawczy
- Skalowalne i autonomiczne wykonywanie procesów - rozproszona wersja silnika JOpery może być uruchomiona na klastrze lub wielordzeniowej maszynie i automatycznie dostosuje się do wykrytej architektury

JOpera jest obecnie rozwijana na Wydziale Informatyki Uniwersytetu Lugano, w Szwajcarii.

2.4.12 Activiti

Activiti¹⁷ jest platformą do zarządzania procesami biznesowymi i scenariuszami. Activiti składa się z kilku współpracujących ze sobą narzędzi:

- Activiti Engine

Główny komponent projektu Activiti, będący silnikiem procesów napisanym w języku Java, który uruchamia natywne procesy BPMN 2.0. Może być uruchomiony w praktycznie każdym środowisku Java - JPA, Spring, jako osobny kontener z obsługą transakcji, oferuje możliwość dodawania nowych, zdefiniowanych przez użytkownika typów aktywności i dedykowanych języków opisu procesów. Wprowadza rozdział technicznych detali na temat oprogramowania procesu od diagramu szkieletu biznesowego. Ponadto umożliwia testowanie wykonania procesów w izolacji, pod postacią zwykłych testów jednostkowych.

- Process Virtual Machine

¹⁷<http://activiti.org>

Jedna z podstawowych warstw architektonicznych Silnika. Pozwala na łatwe ładowanie nowych typów aktywności, cech lub kompletnych języków opisu procesów.

- Activiti Explorer

Aplikacja webowa zapewniająca dostęp do Silnika. Oferuje zarządzanie zadaniami, przeglądanie raportów bazujących na historycznych danych statystycznych i na inspekcje instancji procesów.

- Activiti Probe

Aplikacja webowa pozwalająca na administrację i monitorowanie Silnika, kierowana do administratorów systemowych i operatorów odpowiedzialnych za utrzymanie infrastruktury systemowej.

- Activiti Modeler

Narzędzie do modelowania procesów opierające się na silniku Signavio, oferujące graficzne tworzenie procesów BPMN 2.0.

2.4.13 Bonitasoft

Bonitasoft¹⁸ jest platformą do zarządzania procesami biznesowymi i scenariuszami, składającą się z trzech głównych modułów: Edytora do modelowania procesów, Silnika Wykonawczego i Interfejsu Użytkownika.

Edytor pozwala na graficzną definicję procesów, zależności między ich elementami składowymi, użytkowników końcowych oraz dane wejściowe.

2.4.14 Joget Workflow

Joget Workflow¹⁹ jest narzędziem do wytwarzania scenariuszy i aplikacji BPM (ang. Business Process Management). Posiada rozszerzalną architekturę (system pluginów) pozwalającą na łatwe modyfikowanie aplikacji i dodawanie nowych nowych funkcjonalności. Jednym z jego modułów jest edytor umożliwiający użytkownikowi graficzne modelowanie procesów i scenariuszy.

¹⁸<http://www.bonitasoft.com/>

¹⁹<http://www.joget.org/>

Aplikacja napisana jest w języku Java, co powoduje, że jest niezależna od systemu operacyjnego. Wspiera również szeroką gamę silników baz danych, a dzięki wykorzystaniu technologii JSON i biblioteki JavaScript API jest łatwo integrowalne z innymi technologiami webowymi (PHP, .NET).

Narzędzie rozwijane jest w również w wersji Open Source pod licencją GPL, w okrojonej postaci w stosunku do licencji komercyjnej. Joget Workflow rozwijany jest przez firmę Open Dynamics.

2.4.15 ProcessMaker

ProcessMaker²⁰ jest systemem do zarządzania scenariuszami, pozwalającym firmom i organizacjom na zautomatyzowanie procesów przepływu danych pomiędzy działami i systemami. Umożliwia tworzenie i wykonywanie scenariuszy osobom bez wiedzy programistycznej - biznesmenom i ekspertom od procesów.

²⁰<http://www.processmaker.com/>

Rozdział 3

Kontekst systemu „Kernel Hive”

3.1 Krótkie streszczenie rozdziału

Celem pracy jest zaprojektowanie i zaimplementowanie graficznego narzędzia, pozwalającego w łatwy i intuicyjny sposób na tworzenie scenariuszy, składających się z zadań obliczeniowych oraz relacji czasowych między nimi. Samoistnie istnienie takiej aplikacji pozbawione jest jednak sensu biznesowego, dlatego ww. program jest rozwijany w ramach większego projektu „Kernel Hive”. Ten rozdział ma na celu zaprezentowanie systemu jako spójnej całości, z którą graficzne narzędzie wspomagające projektowanie scenariuszy obliczeniowych jest ściśle powiązane.

3.2 Opis systemu „Kernel Hive”

„Kernel Hive” jest systemem rozwijanym na Katedrze Architektury Systemów Komputerowych Wydziału Elektroniki, Telekomunikacji i Informatyki Politechniki Gdańskiej. Za jego rozwój odpowiedzialny jest zespół projektowy pod przewodnictwem dra inż. Pawła Czarnula. „Kernel Hive” jest systemem oferującym użytkownikowi możliwość łatwego tworzenia równoległych aplikacji rozproszonych a następnie uruchamianie ich w prosty sposób, wykorzystując do obliczeń fizyczne komponenty podłączone do systemu.

Tworzenie aplikacji polega na definiowaniu scenariuszy, składających się z zadań obliczeniowych oraz zależności czasowych między nimi. Użytkownik, mając do dyspozycji graficzne narzędzie wspomagające, projektuje scenariusz reprezentowany jako graf, w którym węzły są zadaniami obliczeniowymi, a krawędzie - zależnościami między nimi. System zwalnia użytkownika ze znajomości projektowania aplikacji rozproszonych i dostarcza zbiór szablonów zadań obliczeniowych. Takie

szablony zawierają już w sobie funkcje realizujące komunikację i wymianę danych między sobą, pozostawiając użytkownikowi jedynie ciało metody realizującej główne zadanie danego szablonu, na przykład podział lub scalenie danych. W ten sposób użytkownik może skupić się na logice biznesowej aplikacji i nie zajmować się niskopoziomowymi szczegółami komunikacyjnymi.

Uruchamianie aplikacji polega na przesłaniu jej do Silnika Wykonawczego systemu, który ma za zadanie umieszczenie scenariusza w kolejce wykonania, następnie jego optymalizację i ostatecznie wykonanie, korzystając z mocy obliczeniowej komponentów fizycznych podłączonych do systemu, takich jak klastry i ich węzły. Celem pełniejszego wykorzystania możliwości węzłów klastrów podłączonych do systemu, możliwe jest użycie do obliczeń zarówno procesorów CPU (ang. *Central Processing Unit*), jak i procesorów graficznych GPU (ang. *Graphical Processing Unit*).

3.3 Architektura systemu „Kernel Hive”

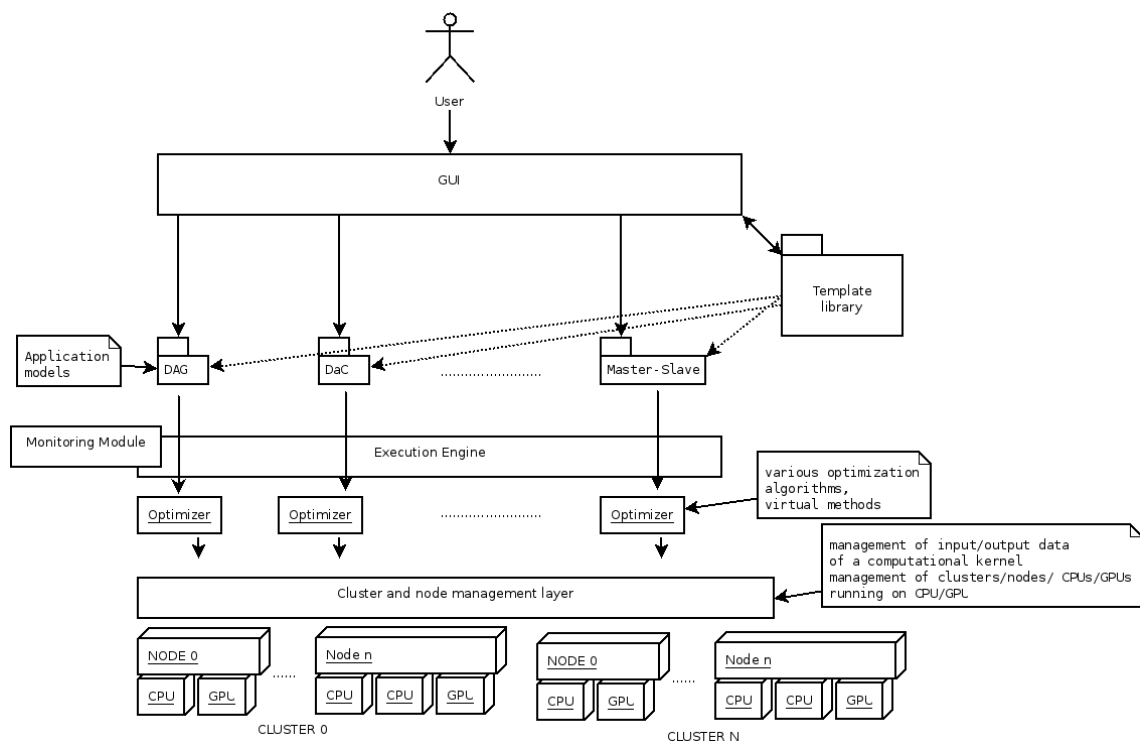
System „Kernel Hive” posiada architekturę wielowarstwową, można jednak wyróżnić tutaj trzy podstawowe warstwy:

- Warstwa Modelu Aplikacji (ang. *Application Model Layer*)
- Warstwa Silnika Wykonawczego (ang. *Execution Engine Layer*)
- Warstwa Zarządzająca Klastrami i Węzłami (ang. *Cluster and Node Management Layer*)

Warstwa Modelu Aplikacji umożliwia modelowanie aplikacji równoległej pod postacią grafu, gdzie jego węzły reprezentują zadania obliczeniowe, a krawędzie - zależności czasowe pomiędzy nimi.

Warstwa Silnika Wykonawczego ma za zadanie umieszczenie aplikacji w kolejce wykonania oraz, wykorzystując podsystem optymalizatorów, przeprowadzenie mapowania poszczególnych zadań obliczeniowych aplikacji równoległej na klastry i ich węzły dostępne w systemie.

Warstwa Zarządzająca Klastrami i Węzłami dostarcza zunifikowane API dla sieci klastrów posiadających węzły obliczeniowe; każdy węzeł posiada natomiast procesor/y CPU i ewentualnie procesor/y GPU.



Rysunek 3.1: Diagram architektury systemu

3.4 Komponenty systemu „Kernel Hive”

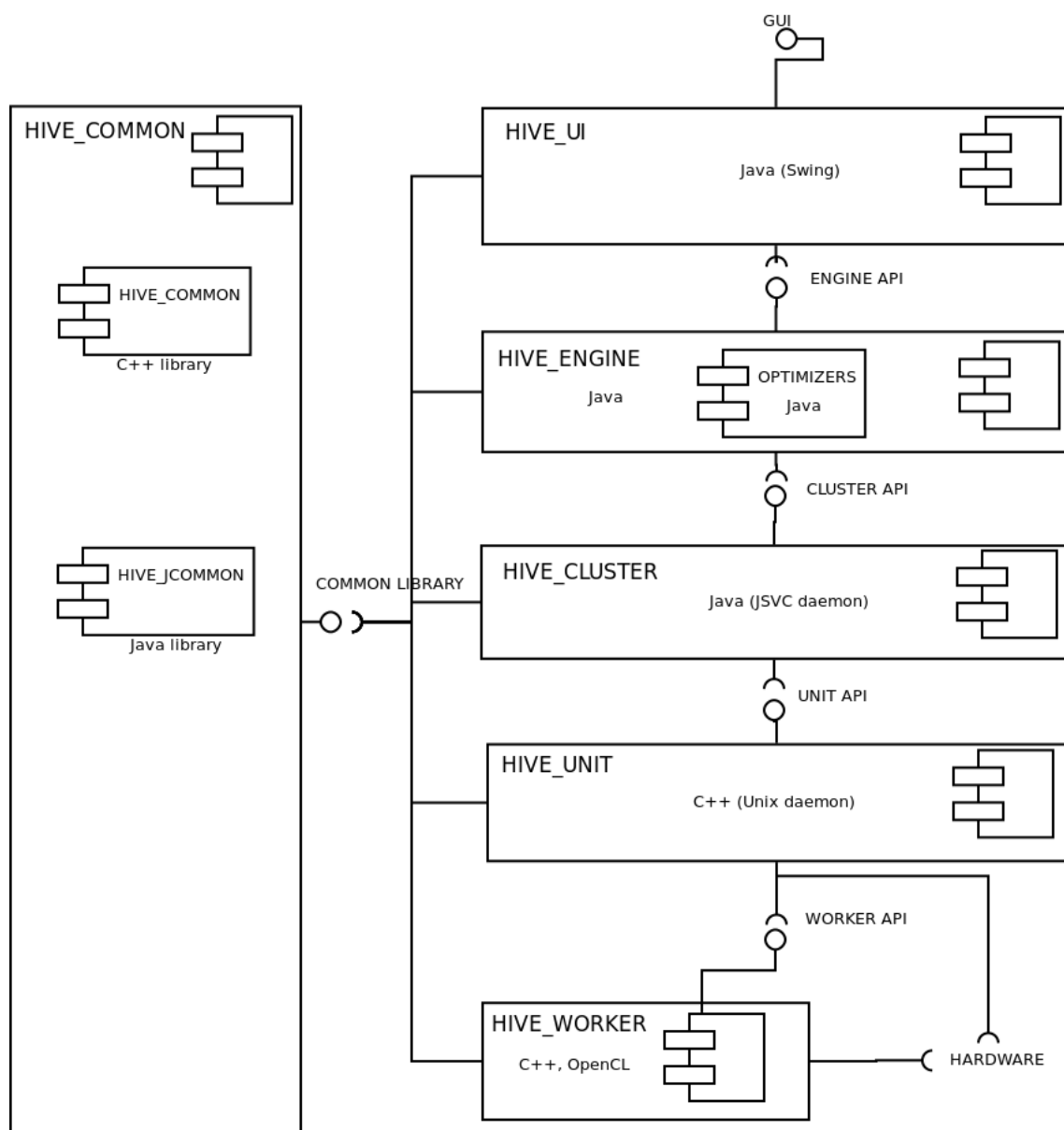
W systemie „Kernel Hive” można wyróżnić 6 głównych komponentów, każdy odpowiedzialny za istotną część systemu. Dzięki zachowaniu architektury wielowarstwowej nie są ze sobą ściśle związane - przerwanie działania jednego z nich nie powoduje zatrzymania systemu, jak również wymiana jednego komponentu nie wymaga zmiany innych.

Komponenty obecne w systemie „Kernel Hive”:

- HIVE_UI - komponent odpowiedzialny za udostępnienie użytkownikowi interfejsu do wykonywania operacji na narzędziu wspomagającym projektowanie rozproszonych aplikacji równoległych. Komponent HIVE_UI jest tematem niniejszej pracy magisterskiej
- HIVE_ENGINE - komponent Silnika Wykonawczego systemu
- HIVE_CLUSTER - komponent fizycznie umieszczony na węzłach dostępowych klastrów podłączonych do systemu. Odpowiada za przyjmowanie zadań obliczeniowych od Silnika Wykonawczego do wykonania na klastrze oraz zarządza jego węzłami
- HIVE_UNIT - komponent fizycznie umieszczony na węzłach obliczeniowych klastrów. Odpowiada za przyjmowanie zadań obliczeniowych od komponentu HIVE_CLUSTER i urucho-

mienie ich odpowiednio

- HIVE_WORKER - komponent odpowiedzialny za prawidłowe wykonanie zadania obliczeniowego. Jest uruchamiany przez komponent HIVE_UNIT
- HIVE_COMMON - komponent odpowiedzialny za przechowywanie zbioru szablonów zadań obliczeniowych. Pełni również funkcję zewnętrznej biblioteki, zawierającej w sobie funkcje współdzielone przez różne komponenty systemu



Rysunek 3.2: Diagram komponentów systemu

Rozdział 4

Specyfikacja wymagań systemowych

4.1 Krótkie streszczenie rozdziału

Wstępne określenie założeń dotyczących systemu jest jednym z najważniejszych etapów tworzenia oprogramowania. Podczas tego procesu definiuje się wymagania użytkownika co do systemu oraz określa główne przypadki użycia w ustandaryzowany sposób. Powstały w ten sposób dokument powinien opisywać wszystkie przewidywane zależności między zdefiniowanymi komponentami systemu.

4.2 Specyfikacja wymagań systemowych

4.2.1 Udziałowcy

STKH_001	Katedra ASK
Opis:	Katedra Architektury Systemów Komputerowych
Typ:	Instytucjonalny
Pełna nazwa:	Katedra Architektury Systemów Komputerowych
Adres:	Politechnika Gdańska, ul. Narutowicza 11/12, 80-233 Gdańsk
Reprezentant:	Dr inż. Paweł Czarnul
Publikacja:	
Priorytet:	Wysoki

4.2.2 Cele systemu

Cele biznesowe

BSGL_001	Stworzenie systemu zarządzania scenariuszami
Opis:	Celem biznesowym jest stworzenie aplikacji umożliwiającej proste i intuicyjne zarządzanie i tworzenie scenariuszy (ang. <i>workflow</i>)
Źródło:	STKH_001 Katedra ASK
Priorytet:	Wysoki

Cele funkcjonalne

FNGL_001	Tworzenie i edytowanie scenariuszy
Opis:	Aplikacja ma na celu umożliwienie Użytkownikowi stworzenie i edytowanie scenariusza obliczeniowego
Źródło:	STKH_001 Katedra ASK
Priorytet:	Wysoki

FNGL_002	Uruchomienie scenariusza
Opis:	Scenariusz zostanie przesłany do Silnika Wykonawczego, a następnie tam uruchomiony i wykonany
Źródło:	STKH_001 Katedra ASK
Priorytet:	Wysoki

FNGL_003	Monitorowanie systemu Silnika Wykonawczego
Opis:	Moduł monitorujący ma za zadanie zbieranie stałych informacji o systemie „Kernel Hive” (ilość i konfiguracja sprzętowa węzłów)
Źródło:	STKH_001 Katedra ASK
Priorytet:	Wysoki

4.2.3 Otoczenie systemu

Użytkownicy

USER_001	Użytkownik
Opis:	Zwykły użytkownik systemu.
Potrzeby:	Tworzenie scenariuszy obliczeniowych
Zadania:	Tworzy i edytuje scenariusze obliczeniowe, a następnie zleca je do wykonania
Źródło:	STKH_001 Katedra ASK
Priorytet:	Wysoki

USER_002	Administrator
Opis:	Administrator systemu wykonawczego
Potrzeby:	Monitorowanie i analiza działania systemu wykonawczego
Zadania:	Opieka nad systemem wykonawczym
Źródło:	STKH_001 Katedra ASK
Priorytet:	Średni

Systemy zewnętrzne

XSYS_001	System Wykonawczy
Opis:	System wykonawczy uruchamia zadane przez użytkowników scenariusze
Potrzeby:	Kod źródłowy scenariusza
Zadania:	Uruchamia scenariusze, rezerwuje zasoby dla ich wykonania
Interfejsy:	Web Services
Źródło:	STKH_001 Katedra ASK
Priorytet:	Wysoki

XSYS_002	Repozytorium Komponentów Scenariuszy
Opis:	Repozytorium z gotowymi komponentami programowymi wielokrotnego użytku, z predefiniowanymi podstawowymi funkcjonalnościami
Potrzeby:	URL szukanego komponentu, zapytanie do wyszukania interesujących nas komponentów
Zadania:	Przechowywanie i udostępnianie komponentów scenariuszy
Interfejsy:	Web Services
Źródło:	STKH_001 Katedra ASK
Priorytet:	Wysoki

XSYS_003	System Operacyjny
Opis:	System operacyjny, pod którym będzie działać aplikacja
Potrzeby:	
Zadania:	Udostępnianie zasobów wymaganych do działania aplikacji
Interfejsy:	Biblioteki systemowe
Źródło:	STKH_001 Katedra ASK
Priorytet:	

XSYS_004	Sieć lokalna
Opis:	Sieć lokalna, używana do połączeń z repozytorium komponentów i z systemem wykonawczym
Potrzeby:	
Zadania:	Łączność z maszynami i usługami w sieci
Interfejsy:	
Źródło:	STKH_001 Katedra ASK
Priorytet:	

4.2.4 Przewidywane komponenty systemu

Komponenty sprzętowe

HCMP_001	Komputer klasy PC
Opis:	Komputer klasy PC, zdolny do uruchomienia aplikacji
Powiązania:	XSYS_003 System Operacyjny, XSYS_004 Sieć Lokalna
Źródło:	STKH_001 Katedra ASK
Priorytet:	

Komponenty programowe

SCMP_001	Pobieranie komponentów scenariuszy z Repozytorium
Opis:	Podsystem komunikacji z Repozytorium Komponentów Scenariuszy
Powiązania:	XSYS_002 Repozytorium Komponentów Scenariuszy
Źródło:	STKH_011 Katedra ASK
Priorytet:	Wysoki

SCMP_002	Tworzenie i edycja scenariuszy
Opis:	Podsystem tworzenia i edycji scenariuszy
Powiązania:	XSYS_001 System Wykonawczy
Źródło:	STKH_011 Katedra ASK
Priorytet:	Wysoki

SCMP_003	Monitorowanie i analiza stanu Systemu Wykonawczego
Opis:	Podsystem monitorowania i analizy stanu Systemu Wykonawczego
Powiązania:	XSYS_001 System Wykonawczy
Źródło:	STKH_011 Katedra ASK
Priorytet:	Wysoki

SCMP_004	Uruchamianie scenariusza
Opis:	Podsystem komunikacji z Systemem Wykonawczym, służący do zlecenia wykonania scenariuszy
Powiązania:	XSYS_001 System Wykonawczy
Źródło:	STKH_011 Katedra ASK
Priorytet:	Wysoki

4.2.5 Wymagania

Wymagania funkcjonalne

FNRQ_001	Tworzenie szkieletu scenariusza z predefiniowalnych, modyfikowalnych komponentów
Opis:	Tworzenie szkieletu scenariusza z predefiniowalnych, modyfikowalnych komponentów
Dotyczy:	SCPM_002 Tworzenie i edycja scenariuszy
Źródło:	STKH_001 Katedra ASK
Powiązania:	XSYS_002 Repozytorium Komponentów Scenariuszy
Priorytet:	Wysoki

FNRQ_002	Odciążenie użytkownika od szczegółów implementacyjnych i komunikacyjnych
Opis:	Pozwala to skupić się jemu na logice biznesowej
Dotyczy:	SCPM_002 Tworzenie i edycja scenariuszy
Źródło:	STKH_001 Katedra ASK
Powiązania:	XSYS_002 Repozytorium Komponentów Scenariuszy
Priorytet:	Wysoki

FNRQ_003	Monitorowanie stanu Systemu Wykonawczego
Opis:	Aplikacja ma mieć możliwość monitorowania stanu ogólnego Systemu Wykonawczego i szczegółowego (jego komponentów)
Dotyczy:	SCMP_003 Monitorowanie i analiza stanu Systemu Wykonawczego
Źródło:	STKH_001 Katedra ASK
Powiązania:	XSYS_001 System Wykonawczy
Priorytet:	Wysoki

Wymagania na dane

DTRQ_001	Komponenty scenariuszowe
Opis:	Komponenty scenariuszowe przechowywane w Repozytorium
Powiązania:	XSYS_002 Repozytorium Komponentów Scenariuszy
Źródło:	STKH_001 Katedra ASK
Priorytet:	Wysoki

DTRQ_001	Informacje o stanie Systemu Wykonawczego
Opis:	Informacje o stanie i konfiguracji Systemu Wykonawczego
Powiązania:	XSYS_001 System Wykonawczy
Źródło:	STKH_001 Katedra ASK
Priorytet:	Wysoki

Wymagania jakościowe

Wymagania w zakresie wiarygodności

RLRQ_001	Stabilność
Opis:	Czas bezawaryjnej pracy aplikacji powinien być równy czasowi jej pracy
Powiązania:	
Źródło:	STKH_001 Katedra ASK
Priorytet:	Wysoki

RLRQ_002	Ochrona zasobów Systemu Wykonawczego
Opis:	Aplikacja powinna uniemożliwić użytkownikowi świadome lub nieświadome naruszenie zasobów Systemu Wykonawczego
Powiązania:	XSYS_001 System Wykonawczy
Źródło:	STKH_001 Katedra ASK
Priorytet:	Wysoki

Wymagania w zakresie wydajności

PFRQ_001	Średnie wymagania sprzętowe
Opis:	Aplikacja powinna dać się uruchomić na średniej klasy komputerze PC
Powiązania:	HCMP_001 Komputer klasy PC
Źródło:	STKH_001 Katedra ASK
Priorytet:	Średni

Wymagania w zakresie elastyczności

FLRQ_001	Aplikacja działająca na systemach z rodziny Linux
Opis:	Aplikacja działająca na systemach z rodziny Linux
Powiązania:	XSYS_003 System operacyjny
Źródło:	STKH_001 Katedra ASK
Priorytet:	Średni

Wymagania w zakresie użyteczności

STRQ_001	Łatwe i intuicyjne tworzenie scenariuszy
Opis:	Aplikacja ma dawać użytkownikowi możliwość łatwego i intuicyjnego tworzenia scenariuszy obliczeniowych
Powiązania:	SCMP_002 Tworzenie i edycja scenariuszy
Źródło:	STKH_001 Katedra ASK
Priorytet:	Wysoki

4.2.6 Sytuacje wyjątkowe

4.2.7 Dodatkowe wymagania

Wymagania sprzętowe

XHRQ_001	Konfiguracja sprzętowa stacji roboczej
Opis:	Stacja robocza musi być wyposażona w kartę sieciową
Dotyczy:	HCMP_001 Komputer klasy PC
Źródło:	STKH_001 Katedra ASK
Priorytet:	Wysoki

Wymagania programowe

XSRQ_001	System operacyjny z rodziny Linux
Opis:	Stacja robocza powinna być wyposażona w system operacyjny z rodziny Linux
Dotyczy:	XSYS_003 System Operacyjny
Źródło:	STKH_001 Katedra ASK
Priorytet:	Średni

XSRQ_001	Biblioteki systemowe potrzebne do uruchomienia aplikacji
Opis:	Biblioteki systemowe potrzebne do uruchomienia aplikacji
Dotyczy:	XSYS_003 System Operacyjny
Źródło:	STKH_001 Katedra ASK
Priorytet:	Wysoki

4.2.8 Kryteria akceptacyjne

ACPT_001	Testy jednostkowe
Opis:	Testy jednostkowe aplikacji
Dotyczy:	SCMP_001 Pobieranie komponentów scenariuszy z Repozytorium, SCMP_002 Tworzenie i edycja scenariuszy, SCMP_003 Monitorowanie i analiza stanu Systemu Wykonawczego, SCMP_004 Uruchamianie scenariusza
Źródło:	STKH_001 Katedra ASK
Priorytet:	Wysoki

ACPT_002	Przejście poznawcze
Opis:	Przejście poznawcze aplikacji przez recenzenta pracy
Dotyczy:	SCMP_001 Pobieranie komponentów scenariuszy z Repozytorium, SCMP_002 Tworzenie i edycja scenariuszy, SCMP_003 Monitorowanie i analiza stanu Systemu Wykonawczego, SCMP_004 Uruchamianie scenariusza
Źródło:	STKH_001 Katedra ASK
Priorytet:	Wysoki

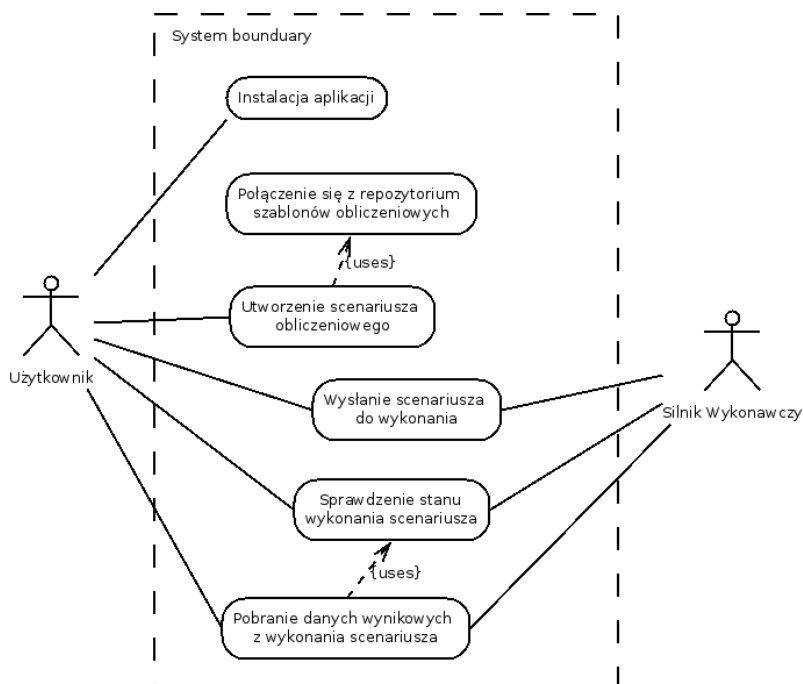
4.2.9 Słownik

- **Zadanie Obliczeniowe** - najmniejsza jednostka programowa, której kod jest wykonywany za pomocą systemu „Kernel Hive”
- **Scenariusz Obliczeniowy** - rozproszona aplikacja w formie scenariusza (ang. *workflow*) składająca się z Zadań Obliczeniowych i relacji czasowych między nimi
- **Narzędzie Wspomagające** - graficzna aplikacja, pozwalająca w łatwy i intuicyjny sposób na tworzenie Scenariuszy, składających się z Zadań Obliczeniowych oraz relacji czasowych między nimi

- **Użytkownik** - osoba korzystająca z Narzędzia Wspomagającego systemu „Kernel Hive”
- **System Wykonawczy** - zbiór komponentów systemu „Kernel Hive”, odpowiadający za uruchomienie i wykonanie Scenariusza Obliczeniowego
- **Silnik Wykonawczy** - element centralny Systemu Wykonawczego, odpowiada za pobieranie Scenariuszy Obliczeniowych, rozdzielanie Zadań Obliczeniowych na węzły sprzętowe dostępne w systemie, zaplanowanie ich wykonania oraz przechowywanie i udostępnianie wyników obliczeń
- **Repozytorium Szablonów Obliczeniowych** - komponent systemu „Kernel Hive” odpowiedzialny za przechowywanie szablonów Zadań Obliczeniowych, które można pobrać i wykorzystać w tworzonym Scenariuszu Obliczeniowym

4.3 Główne przypadki użycia

4.3.1 Diagram głównych przypadków użycia



Rysunek 4.1: Diagram głównych przypadków użycia

4.3.2 Opis głównych przypadków użycia

Użytkownikiem końcowym narzędzia będą osoby ze środowisk naukowych, niekoniecznie o wykształceniu informatycznym. Narzędzie zwalnia ich ze znajomości pisania aplikacji uruchamianych na systemach gridowych, oczekuje jednak wiedzy na temat tworzenia prostych kerneli programowych wykonujących faktyczne obliczenia.

Utworzenie scenariusza obliczeniowego

Aktorzy: Użytkownik

Użytkownik chce utworzyć scenariusz obliczeniowy, który następnie będzie mógł być wykonany po przesłaniu go do silnika wykonawczego.

1. Użytkownik tworzy nowy projekt, zawierający w sobie scenariusz obliczeniowy
2. Aplikacja łączy się z repozytorium szablonów obliczeniowych, z których może skorzystać użytkownik. Szablony implementują zachowania najczęściej spotykanych paradygmatów programowania równoległego: Master-Slave, DaC, Pipeline itp.
3. Użytkownik pobiera wybrany przez siebie szablon i dodaje go do scenariusza
4. Użytkownik określa zależności czasowe między pobranym szablonem a poprzednio dodanymi
5. Użytkownik modyfikuje ciało metody szablonu odpowiedzialnej za obliczenia
6. Użytkownik określa wartości wymaganych parametrów przypisanych do danego szablonu
7. Użytkownik może powtórzyć akcje wykonywane w punktach 3-6

Sytuacje wyjątkowe:

W przypadku niepołączenia się z repozytorium szablonów, aplikacja wyświetli stosowny komunikat o błędzie.

Wysłanie scenariusza do wykonania

Aktorzy: Użytkownik, Silnik Wykonawczy

Użytkownik chce wykonać utworzony scenariusz obliczeniowy, by uzyskać wyniki.

1. Użytkownik inicjuje akcję wysłania scenariusza do Silnika Wykonawczego
2. Aplikacja przeprowadza walidację scenariusza pod kontem poprawności i kompletności jego konstrukcji
3. Użytkownik podaje dane wejściowe, na których będą przeprowadzane obliczenia
4. Użytkownik podaje swój login oraz hasło celem uwierzytelnienia i autoryzacji przez Silnik Wykonawczy
5. Silnik Wykonawczy uwierzytelnia i autoryzuje Użytkownika
6. Silnik Wykonawczy otrzymuje scenariusz oraz jego dane wejściowe
7. Silnik Wykonawczy umieszcza scenariusz w systemie kolejkowym, przypisując mu unikalny identyfikator
8. Silnik Wykonawczy zwraca Użytkownikowi unikalny identyfikator scenariusza celem jego późniejszej identyfikacji

Sytuacje wyjątkowe:

Scenariusz nie przechodzi procesu walidacji. W tym wypadku aplikacja wyświetli stosowny komunikat opisujący te części scenariusza, które spowodowały błąd.

Użytkownik poda błędny login i/lub hasło. W tym wypadku Silnik Wykonawczy poinformuje Użytkownika o nieudanym uwierzytelnieniu. Procedurę wysłania scenariusza do wykonania należy przeprowadzić wówczas jeszcze raz.

Aplikacja nie może połączyć się z Silnikiem Wykonawczym. W tym wypadku aplikacja wyświetli stosowny komunikat o błędzie.

Sprawdzenie stanu wykonania scenariusza

Aktorzy: Użytkownik, Silnik Wykonawczy

Użytkownik chce sprawdzić stan wykonania scenariusza obliczeniowego.

1. Użytkownik inicjuje akcję wyświetlenia listy wykonań jego scenariuszy, podając swój login i hasło

2. Aplikacja łączy się z Silnikiem Wykonawczym i otrzymuje od niego wymagane dane
3. Aplikacja wyświetla listę wykonanych scenariuszy Użytkownika
4. Użytkownik przeszukuje listę, by znaleźć interesujące go wykonanie scenariusza
5. Użytkownik widzi stan wykonania scenariusza oraz inne, związane z nim informacje (komunikaty o błędach, logi, a w przypadku poprawnego wykonania - dane wyjściowe)

Sytuacje wyjątkowe:

Użytkownik poda błędny i/lub hasło. W tym wypadku Silnik Wykonawczy poinformuje Użytkownika o nieudanym uwierzytelnieniu. Procedurę sprawdzenia stanu wykonania scenariusza należy przeprowadzić wówczas jeszcze raz.

Aplikacja nie może połączyć się z Silnikiem Wykonawczym. W tym wypadku aplikacja wyświetli stosowny komunikat o błędzie.

Pobranie danych wynikowych z wykonania scenariusza

Aktorzy: Użytkownik, Silnik Wykonawczy

Użytkownik chce pobrać dane wynikowe z interesującego go wykonania scenariusza.

1. Użytkownik przeprowadza akcję "Sprawdzenie stanu wykonania scenariusza"
2. Jeżeli interesujące go wykonanie scenariusza zostało zakończone pomyślnie, Użytkownik inicjuje akcję pobierania danych wynikowych
3. Aplikacja łączy się z Silnikiem Wykonawczym i pobiera od niego dane wynikowe
4. Aplikacja zapisuje dane wynikowe do pliku wskazanego przez Użytkownika

Sytuacje wyjątkowe:

Aplikacja nie może połączyć się z Silnikiem Wykonawczym. W tym wypadku aplikacja wyświetli stosowny komunikat o błędzie.

Instalacja aplikacji

Aktorzy: Użytkownik

Użytkownik chce zainstalować narzędzie wspomagające projektowanie rozproszonych aplikacji równoległych.

1. Użytkownik pobiera narzędzie w formie archiwum JAR oraz skryptu startowego
2. Użytkownik umieszcza powyższe pliki w wybranym przez siebie miejscu

Rozdział 5

Projekt systemu

5.1 Krótkie streszczenie rozdziału

Projekt jest drugim z najważniejszych etapów tworzenia oprogramowania. Podczas tego procesu dokonuje się analizy wymagań architektonicznych stawianych przez warunki zewnętrzne i klienta. Powstały w ten sposób dokument powinien opisywać wszystkie zależności między zdefiniowanymi komponentami systemu. Projekt architektury systemu jest podstawowym opisem całości systemu, niezbędnym, by przejść do następnego etapu wytwarzania systemu - implementacji.

5.2 Analiza wymagań architektonicznych

5.2.1 Wymagania architektoniczne przypadków użycia

Utworzenie scenariusza obliczeniowego

Wymaga się, aby system umożliwiał utworzenie scenariusza obliczeniowego w sposób łatwy i intuicyjny.

Wykorzystane zostaną wskazówki dotyczące standaryzacji wyglądu i zachowania interfejsów graficznych dla biblioteki Java Swing.

Wysłanie scenariusza do wykonania

System powinien umożliwiać wysłanie scenariusza do Silnika Wykonawczego w nieskomplikowany sposób. Niemożność wysłania scenariusza nie może skutkować przerwaniem działania aplikacji.

Proces wysyłania scenariusza nie może powodować blokady innych funkcji programu.

Wysyłanie scenariusza powinno być zrealizowane za pomocą technologii umożliwiającej zabezpieczenie się przed błędami w komunikacji. Interfejs wykorzystywany do wysłania scenariusza powinien być standardem, co nie będzie generowało problemów przy projektowaniu innej aplikacji realizującej ten przypadek użycia. Przesyłanie scenariusza powinno odbywać się w osobnym wątku programu, tak by nie blokować innych jego funkcji.

Sprawdzenie stanu wykonania scenariusza

System powinien umożliwiać sprawdzenie stanu wykonania scenariusza w nieskomplikowany sposób. Niemożność sprawdzenia stanu wykonania scenariusza nie może skutkować przerwaniem działania aplikacji. Proces sprawdzenia stanu wykonania scenariusza nie może powodować blokady innych funkcji programu.

Sprawdzenie stanu wykonania scenariusza powinno być zrealizowane za pomocą technologii umożliwiającej zabezpieczenie się przed błędami w komunikacji. Interfejs wykorzystywany do sprawdzenia stanu wykonania scenariusza powinien być standardem, co nie będzie generowało problemów przy projektowaniu innej aplikacji realizującej ten przypadek użycia. Sprawdzenie stanu wykonania scenariusza powinno odbywać się w osobnym wątku programu, tak by nie blokować innych jego funkcji.

Pobranie danych wynikowych z wykonania scenariusza

System powinien umożliwiać pobieranie danych wynikowych w nieskomplikowany sposób. Niemożność pobrania danych wynikowych nie może skutkować przerwaniem działania aplikacji. Proces pobierania danych wynikowych nie może powodować blokady innych funkcji programu.

Pobieranie danych wynikowych powinno być zrealizowane za pomocą technologii umożliwiającej zabezpieczenie się przed błędami w komunikacji. Interfejs wykorzystywany do pobierania danych wynikowych powinien być standardem, co nie będzie generowało problemów przy projektowaniu innej aplikacji realizującej ten przypadek użycia. Pobieranie danych wynikowych powinno odbywać się w osobnym wątku programu, tak by nie blokować innych jego funkcji.

Instalacja aplikacji

Aplikacja powinna być dystrybuowana w formie łatwej do instalacji.

Aplikacja powinna być rozpowszechniana w postaci archiwum plików, które następnie należy rozpakować do wybranego przez siebie katalogu.

5.2.2 Wybór przypadków użycia sterujących architekturą systemu

Najbardziej kluczowymi cechami narzędzia są łatwość obsługi, intuicyjność i użyteczność. W związku z powyższym, następujące przypadki użycia są przypadkami sterującymi:

- Utworzenie scenariusza obliczeniowego
- Wysłanie scenariusza do wykonania
- Sprawdzenie stanu wykonania scenariusza
- Pobranie danych wynikowych z wykonania scenariusza

Proces instalacji będzie wykonywany bardzo rzadko i jego przebieg nie jest kluczowy dla architektury (to proces instalacji będzie dostosowany do architektury wynikającej z pozostałych funkcjonalności, a nie odwrotnie).

5.3 Projekt architektury systemu

5.3.1 Koncepcja architektury systemu

System zostanie zrealizowany w postaci aplikacji desktopowej, działającej na komputerze użytkownika. Będzie składał się z 5 głównych podsystemów:

- Podsystemu głównego (main)
- Podsystemu odpowiedzialnego za wyświetlanie i pobieranie szablonów obliczeniowych (repository)
- Podsystemu odpowiedzialnego za wszelkie operacje manipulujące scenariuszem (workflow)
- Podsystemu odpowiedzialnego za realizację komunikacji (networking)
- Podsystemu odpowiedzialnego za operacje wykonywane na grafie - reprezentacji graficznej scenariusza (graph)

5.3.2 Schemat architektury systemu

5.3.3 Komponenty sprzętowe

HCMP_001	Stacja robocza użytkownika
Opis:	Stacja robocza należąca do użytkownika
Powiązania:	

XHRQ_001	Konfiguracja stacji roboczej
Opis:	Stacja robocza posiada zainstalowaną maszynę wirtualną języka Java
Powiązania:	HCMP_001 Stacja robocza użytkownika

5.3.4 Podsystemy

SSYS_001	main
Opis:	Podsystem main odpowiada za uruchomienie systemu, obsługę sygnałów systemowych oraz za obsługę głównego okna aplikacji i akcji z nim związanych
Przypadki użycia:	Uruchomienie aplikacji, zamknięcie aplikacji
Lokalizacja:	HCMP_001 Stacja robocza użytkownika
Komponenty:	Plik główny programu (KernelHiveMain.java), klasa głównego okna (MainFrame)
Powiązania:	Wszystkie pozostałe podsystemy

SSYS_002	configuration
Opis:	Podsystem configuration odpowiada za przechowywanie i modyfikację zmiennych konfiguracyjnych programu
Przypadki użycia:	Uruchomienie aplikacji w konkretnym języku, zmiana zmiennych konfiguracyjnych programu
Lokalizacja:	HCMP_001 Stacja robocza użytkownika
Komponenty:	Plik główny konfiguracji (AppConfiguration.java)
Powiązania:	Wszystkie podsystemy korzystające z konfiguracji programu

SSYS_003	dialog
Opis:	Podsystem dialog odpowiada za obsługę wszystkich okienek dialogowych występujących w aplikacji
Przypadki użycia:	Utworzenie nowego projektu, wyświetlenie powiadomienia, prezentacja właściwości projektu
Lokalizacja:	HCMP_001 Stacja robocza użytkownika
Komponenty:	Pliki z klasami dziedziczącymi po javax.swing.JDialog
Powiązania:	Wszystkie podsystemy korzystające wyświetlania informacji za pomocą okien dialogowych

SSYS_004	repository
Opis:	Podsystem repository odpowiada za wyświetlanie i pobieranie szablonów obliczeniowych zdefiniowanych w repozytorium szablonów obliczeniowych
Przypadki użycia:	Pobranie nowego węzła obliczeniowego celem dodania go do scenariusza
Lokalizacja:	HCMP_001 Stacja robocza użytkownika
Komponenty:	Plik z klasą RepositoryViewer
Powiązania:	Podsystem main, podsystem workflow

SSYS_005	workflow
Opis:	Podsystem workflow odpowiada za wszelkie operacje manipulujące scenariuszem (jego utworzenie, dodanie nowych węzłów obliczeniowych)
Przypadki użycia:	Dodanie nowego węzła obliczeniowego do scenariusza, usunięcie węzła obliczeniowego ze scenariusza
Lokalizacja:	HCMP_001 Stacja robocza użytkownika
Komponenty:	Plik z klasą WorkflowEditor
Powiązania:	Podsystem main, podsystem repository, podsystem graph

SSYS_006	security
Opis:	Podsystem security odpowiada za sprawdzanie i przechowywanie haseł użytkownika celem jego uwierzytelnienia w Silniku Wykonawczym
Przypadki użycia:	Wysłanie scenariusza do wykonania, sprawdzenie statusu wykonania scenariuszy danego użytkownika
Lokalizacja:	HCMP_001 Stacja robocza użytkownika
Komponenty:	PasswordKeyStore, PasswordEncryptionService
Powiązania:	Podsystem networking, podsystem components

SSYS_007	networking
Opis:	Podsystem networking odpowiada za realizację komunikacji między aplikacją a Silnikiem Wykonawczym
Przypadki użycia:	Wysłanie scenariusza do wykonania, sprawdzenie statusu wykonania scenariuszy danego użytkownika, sprawdzenie obciążenia systemu, pobranie informacji na temat konfiguracji systemu
Lokalizacja:	HCMP_001 Stacja robocza użytkownika
Komponenty:	WorkflowService, InfrastructureService, MonitorService
Powiązania:	Podsystem security, podsystem components

SSYS_008	components
Opis:	Podsystem components zawiera w sobie proste komponenty graficzne, wyświetlające istotne dla użytkownika informacje
Przypadki użycia:	Zaprezentowanie drzewa katalogowego projektu, edycja kodu źródłowego metody obliczeniowej danego węzła, zaprezentowanie obecnej architektury systemu, zaprezentowanie aktualnego obciążenie systemu
Lokalizacja:	HCMP_001 Stacja robocza użytkownika
Komponenty:	SourceCodeEditor, FileTree, InfrastructureViewer, MonitorViewer
Powiązania:	Podsystem security

SSYS_009	graph
Opis:	Podsystem graph odpowiada za operacje wykonywane na grafie, będącym reprezentacją scenariusza obliczeniowego
Przypadki użycia:	Dodanie nowego węzła do grafu, połączenie węzłów grafu krawędzią, usunięcie węzła z grafu, zapisanie grafu do pliku, odczytanie grafu z pliku
Lokalizacja:	HCMP_001 Stacja robocza użytkownika
Komponenty:	GraphNode, GraphNodeType, GraphConfiguration
Powiązania:	Podsystem workflow

5.3.5 Warstwy architektoniczne

LAYR_001	Warstwa prezentacji
Opis:	Komunikacja z użytkownikiem za pomocą informacji wyświetlanych w oknie aplikacji

LAYR_002	Warstwa logiki biznesowej
Opis:	Najważniejsza warstwa programu, większość podsystemów dotyczy właśnie tej warstwy

LAYR_002	Warstwa danych
Opis:	Zapisywanie i odczytywanie projektów do/z plików, odczyt/zapis zmiennych konfiguracyjnych, odczyt szablonów obliczeniowych z repozytorium szablonów

5.3.6 Komponenty programowe

SCMP_001	Biblioteka Swing
Opis:	Biblioteka graficzna wykorzystana do zaprojektowania okien aplikacji

SCMP_002	Biblioteka Apache Commons Configuration
Opis:	biblioteka odpowiedzialna za odczyt/zapis z/do plików konfiguracyjnych i plików projektów

5.3.7 Sposób realizacji wymagań jakościowych

Wymagania w zakresie wiarygodności

RLRQ_001	Stabilność
Opis:	Czas bezawaryjnej pracy aplikacji powinien być równy czasowi jej pracy
Sposób realizacji:	Wytworzenie wysokiej jakości oprogramowania, wykorzystując przy tym testy jednostkowe i integracyjne aplikacji
Powiązania:	
Źródło:	STKH_001 Katedra ASK
Priorytet:	Wysoki

RLRQ_002	Ochrona zasobów Systemu Wykonawczego
Opis:	Aplikacja powinna uniemożliwić użytkownikowi świadome lub nieświadome naruszenie zasobów Systemu Wykonawczego
Sposób realizacji:	Uzupełnienie procedury wysyłania danych o etap walidacji (po stronie użytkownika - by uchronić się przed błędem ludzkim, po stronie Silnika - by upewnić się, że dane zostały przesłane poprawnie)
Powiązania:	XSYS_001 System Wykonawczy
Źródło:	STKH_001 Katedra ASK
Priorytet:	Wysoki

Wymagania w zakresie wydajności

PFRQ_001	Średnie wymagania sprzętowe
Opis:	Aplikacja powinna dać się uruchomić na średniej klasy komputerze PC
Sposób realizacji:	Rozważne wykorzystanie dostępnych zasobów, zaprojektowanie interfejsu spójnego i logicznego, bez zbędnych graficznych efektów
Powiązania:	HCMP_001 Komputer klasy PC
Źródło:	STKH_001 Katedra ASK
Priorytet:	Średni

Wymagania w zakresie elastyczności

FLRQ_001	Aplikacja działająca na systemach z rodziny Linux
Opis:	Aplikacja działająca na systemach z rodziny Linux
Sposób realizacji:	Aplikacja będzie napisana w języku Java - będzie działać nie tylko na komputerach z systemem z rodziny Linux, ale również na innych
Powiązania:	XSYS_003 System operacyjny
Źródło:	STKH_001 Katedra ASK
Priorytet:	Średni

Wymagania w zakresie użyteczności

STRQ_001	Łatwe i intuicyjne tworzenie scenariuszy
Opis:	Aplikacja ma dawać użytkownikowi możliwość łatwego i intuicyjnego tworzenia scenariuszy obliczeniowych
Sposób realizacji:	Zaprojektowanie kontrolki, pozwalającej na graficzną edycję grafu reprezentującego scenariusz obliczeniowy
Powiązania:	SCMP_002 Tworzenie i edycja scenariuszy
Źródło:	STKH_001 Katedra ASK
Priorytet:	Wysoki

Rozdział 6

Problemy implementacyjne

6.1 Krótkie streszczenie rozdziału

Implementacja jest kolejnym po projektowaniu krokiem w cyklu wytwarzania oprogramowania. W tej części pracy skupiono się na metodologii projektowania i implementacji, środowiskach i narzędziach wspomagających pracę wykorzystanych w trakcie tego etapu, napotkanych problemach implementacyjnych i koncepcyjnych oraz przedstawiono użyte rozwiązania powyższych problemów. Oprócz tego zawarto również fragmenty kodu oraz opisy najważniejszych części architektury systemu.

6.2 Metodologia projektowania i implementacji

6.2.1 Metodologia projektowania systemu

Wybór odpowiedniej metodologii projektowania podlegał warunkom nakładanym przez sam system. Zdecydowano się skorzystać tu z modelu przyrostowego, w odniesieniu do całości systemu „Kernel Hive”, jak i do każdego z jego elementów. Ten wybór podyktowany został specyfiką współpracy z Klientem (Promotorem). Można w tym wypadku wyróżnić poszczególne fazy projektowania:

1. Na początku skupiono się całkowicie na jak najdokładniejszym określeniu całości wymagań i uwspólnieniu wizji systemu tak, by zminimalizować konieczność ewentualnych poprawek w architekturze systemu podczas fazy implementacji
2. Wybrano podzbiór funkcji systemu

3. Wykonano szczegółowy projekt komponentów spełniających zadane funkcje i je zaimplementowano
4. Wykonano testy integracyjne komponentów i zaprezentowano wyniki Promotorowi
5. Powtarzano fazy 2 - 5 celem implementacji całości systemu

6.2.2 Metodologia implementacji

W trakcie fazy implementacji zdecydowano się na zastosowanie techniki TTD (ang. *Test-driven Development*), zaliczaną do metodyk zwinnych (ang. *Agile*). Jej wykorzystanie polega na wielokrotnym powtarzaniu poniższych kroków:

1. Programista pisze test jednostkowy sprawdzający dodawaną funkcjonalność
2. Programista implementuje funkcjonalność, jednocześnie testując jej wykonanie za pomocą wcześniej napisanego testu
3. Programista dokonuje refaktoryzacji kodu tak, by spełniał oczekiwane standardy.

TTD jako metodologia implementacji opierająca się na testach jednostkowych wprowadza szereg ich zalet:

- Natychmiastowe sprawdzenie poprawności kodu - nie trzeba uruchamiać całego programu, by sprawdzić jedną funkcjonalność
- Prowadzi do lepszego designu kodu - stosowanie testów jednostkowych zwiększa niezależność poszczególnych komponentów programu
- Opisuje, w jaki sposób działa program - każdy z testów jednostkowych opisuje zachowanie systemu w konkretnym przypadku dla określonych danych wejściowych
- Ułatwia refaktoryzację - testy jednostkowe sprawiają, że znika obawa przed zmianami w systemie, które mogą prowadzić do zepsucia kodu - testy natychmiast to wykryją

6.3 Środowiska i narzędzia wspomagające

Podczas projektowania usługi wykorzystano szereg narzędzi, wspomagających zarówno wytwarzanie kodu jak również pracę zespołową.

Jako środowisko bazowe wybrano system operacyjny z rodziny GNU/Linux - dystrybucję dystrybucję Fedora. Aplikację przetestowano również na dystrybucji Debian i Ubuntu - taki wybór miał zapewnić przekrojowe testowanie programu na systemie GNU/Linux.

W pracy wykorzystano następujące narzędzia:

- Środowisko programistyczne Eclipse

Zintegrowane środowisko programistyczne udostępniające usługę wielu języków programowania, najczęściej używane przy tworzeniu projektów w języku Java. Wspomaga proces wytwórczy poprzez automatyczną generację skryptów budujących Ant, umożliwia graficzne projektowanie okien aplikacji, wspiera proces testowania jednostkowego oraz posiada obsługę systemów kontroli wersji. Dodatkowo, jest w pełni rozszerzalne dzięki rozbudowanemu systemowi wtyczek.

- JUnit Test Framework

Framework wspomagający programistę przy pisaniu testów jednostkowych dla klas w języku Java, w pełni integruje się z Eclipse. Ponadto realizuje oddzielenie testów od kodu programu, oferuje wiele mechanizmów uruchamiania i udostępnia funkcję tworzenia raportów z przebiegu testów.

- SVN (Subversion)

Do wspomaganie pracy zespołowej wykorzystano narzędzie kontroli wersji Subversion. Doskonale wspomaga pracę przy projekcie grupowym, udostępniając mechanizmy tworzenia lokalnych wersji programu oraz scalania ich na serwerze SVN. Subversion jest jedną z usług udostępnianych przez serwer uczelniany Katedry Architektury Systemów Komputerowych - tam też utworzono repozytorium kodu.

- Trac

Trac jest otwartym systemem zarządzania projektem i bugtrackerem. Pozwala utrzymać kontrolę nad szybko rozrastającym się kodem aplikacji oraz monitoruje programistę o zgłoszonych przez użytkowników i testerów problemach. Narzędzie Trac jest usługą również udostępnianą przez serwer uczelniany Katedry Architektury Systemów Komputerowych, który integruje go z repozytoriami SVN.

- Apache Ant

Apache Ant jest narzędziem do automatyzacji procesu kompilacji, testowania i budowy oprogramowania. Jest zazwyczaj używany przy projektach pisanych w języku Java. Konfiguracja narzędzia odbywa się za pomocą pliku w formacie XML, określającego możliwe cele do wykonania (np. wyczyść katalog wyjściowy, testuj, kompiluj, buduj) oraz akcje, które narzędzie musi wykonać by je osiągnąć.

6.4 Główne problemy implementacyjne

W trakcie rozwoju usługi napotkano na różne problemy natury informatycznej, zarówno implementacyjne, jak i koncepcyjne. Poniżej opisano główne problemy napotkane w trakcie wytwarzania aplikacji.

Serializacja i deserializacja scenariusza

Wybór odpowiedniego sposobu serializacji i deserializacji scenariusza był jedną z bardziej czasochłonnych decyzji w trakcie implementacji projektu. Należało rozważyć tu kilka ważnych aspektów. Jednym z nich był sposób zapisywania scenariusza tak, by przy następnym uruchomieniu aplikacji można go było ponownie otworzyć. Zastanowienia wymagał również sposób zapisywania scenariusza do formy akceptowanej przez Silnik Wykonawczy (część danych zawartych w pliku z zapisanym scenariuszem była wymagana jedynie przez Narzędzie Wspomagające - Silnik ich nie potrzebował) i odpowiedniej do przesłania do niego przez wybrany protokół komunikacyjny.

Komunikacja z Silnikiem Wykonawczym

Prawidłowe rozwiązanie problemu komunikacji narzędzia z Silnikiem Wykonawczym było jednym z ważniejszych zadań w trakcie pracy nad projektem. Wymagane było stworzenie architektury wielowątkowej i odpornej na błędy w trakcie przesyłania danych. Ponieważ po zakończeniu prac zamierzano udostępnić kod źródłowy całego systemu pod licencją GPL, wykorzystany protokół komunikacyjny powinien być ogólnie przyjętym, niezależnym od języka programowania standardem, by umożliwić społeczności Open Source na swobodne dodawanie nowych elementów systemu, korzystając z ustanowionego protokołu.

Komunikacja z Repozytorium Szablonów Obliczeniowych

Rozwiązanie kwestii komunikacji z Repozytorium Szablonów Obliczeniowych było również ważnym etapem implementacji projektu. Wymaganie, by Repozytorium było osobnym podsystemem, wyraźnie oddzielnym od reszty komponentów aplikacji było nad wyraz słuszne - oczywistym było,

że dodanie nowego typu szablonu obliczeniowego nie może powodować konieczności przebudowania całego narzędzia. Co więcej, analiza schematu działania systemu wykazała, że obowiązkowe jest uwspólnienie wersji Repozytorium dla narzędzia wspomagającego i dla Silnika Wykonawczego - niedopuszczalnym jest scenariusz, w którym narzędzie odnosi się do szablonu, którego Silnik nie rozpozna.

Edycja kodu źródłowego metod obliczeniowych

Jedną z głównych funkcji aplikacji jest możliwość edycji metod obliczeniowych szablonów. Ciąła tych metod często są bardzo skomplikowane, a co za tym idzie, duże. Konieczne było jak największe ułatwienie użytkownikowi pracy nad tak długim kodem, oferując edytor o możliwościach zbliżonych do tych obecnych w zintegrowanych środowiskach programistycznych dostępnych na rynku. Domykanie nawiasów, wykrywanie nawiasu parującego obecnie zaznaczony, zwijanie części kodu czy podświetlanie składni to tylko niektóre z funkcji, które powinny znaleźć się w takim edytorze.

Sposób tworzenia scenariuszy

Drugą z głównych funkcji aplikacji jest możliwość łatwego i intuicyjnego tworzenia scenariuszy obliczeniowych przez użytkownika. By spełnić te dwa wymogi, należało maksymalnie uprościć proces tworzenia nowego scenariusza, dodawania nowych zadań obliczeniowych i edycji już istniejących. Ponieważ scenariusz miał być reprezentowany za pomocą acyklicznego grafu skierowanego, w którym węzły oznaczałyby zadania obliczeniowe, a krawędzie - zależności czasowe między nimi, oznaczało to, że należało wybrać odpowiednio intuicyjny opis takiego grafu.

6.5 Przykłady rozwiązań szczegółowych

6.5.1 Sposób serializacji i deserializacji scenariusza obliczeniowego

Określenie koncepcji serializacji i deserializacji scenariusza obliczeniowego wymagało zastanowienia się nad paroma wymaganiami.

Po pierwsze, wiadomym było, że proces serializacji będzie odbywał się podczas zapisywania projektu na dysk tak, by podczas kolejnego uruchomienia aplikacji można było wczytać (deserializować) go poprawnie. W tym kontekście poprawność oznacza wczytanie kompletu informacji na temat scenariusza i sposobu jego prezentacji użytkownikowi.

Po drugie, miano na uwadze, że podobny proces będzie przebiegał podczas wysyłania scenariusza do wykonania. W tym wypadku scenariusz będzie serializowany do postaci możliwej do przesłania wybranym kanałem komunikacyjnym do Silnika Wykonawczego, tam zaś nastąpi poprawna deserializacja. W tym kontekście poprawność tego procesu oznacza wczytanie wszelkich informacji (i tylko nich), które wymagane są do wykonania danego scenariusza.

Należy tu zwrócić uwagę na różnice w wymaganiach co do tych dwóch przypadków. W pierwszym, aplikacja musi przechować również dane o sposobie prezentacji danego scenariusza, w drugim zaś nie jest to ani konieczne, ani nawet pożądane (nadmiarowość przesyłanych danych).

Kolejnym warunkiem sterującym sposobem serializacji i deserializacji scenariusza był sposób zapisu ciał metod obliczeniowych poszczególnych zadań. Po stronie klienta metody te trzymane są w osobnych plikach, utrzymując w ten sposób strukturę katalogową projektu, natomiast po stronie Silnika Wykonawczego scenariusze i metody obliczeniowe ich zadań trzymane są w pamięci serwera.

Z powyższych rozważań wynika, że ten sam scenariusz jest różnie opisywany po stronie Narzędzia Wspomagającego i Silnika Wykonawczego. Możliwym jest jednak wydzielenie zbioru elementów struktury, które są takie same po obydwu stronach. Pierwszym krokiem do poprawnej implementacji procesu serializacji/deserializacji było zaprojektowanie takiej struktury opisującej scenariusz, która wzięłaby powyższe wnioski pod uwagę.

Listing 6.1: Struktura opisująca elementy wspólne scenariusza

```
public enum GraphNodeType {
    GENERIC( "generic" ),
    MERGER( "DataMerger" ),
    PARTITIONER( "DataPartitioner" ),
    PROCESSOR( "DataProcessor" ),
    COMPOSITE( "composite" ),
    MASTERSLAVE( "masterslave" ),
    DAC( "dac" );

    (...)
}

public interface IGraphNode {

    String getId();
    String getName();
    GraphNodeType getType();

    List<IGraphNode> getFollowingNodes();
}
```

```
List<IGraphNode> getPreviousNodes();

IGraphNode getParentNode();

List<IGraphNode> getChildrenNodes();

Map<String, Object> getProperties();

List<ValidationResult> validate();

(...)
}

public class ProcessorGraphNode implements IGraphNode, Serializable {
    (...)
}

public class PartitionerGraphNode implements IGraphNode, Serializable {
    (...)
}

public class MergerGraphNode implements IGraphNode, Serializable {
    (...)
}

public abstract class AbstractGraphNodeDecorator {

    protected IGraphNode node;

    (...)

    public IGraphNode getGraphNode() {
        return node;
    }

    public abstract List<ValidationResult> validate();
}
```

Jak można dostrzec na powyższym listingu, zadania scenariusza obliczeniowego przyjmują tu postać węzłów grafu (interfejs `IGraphNode`), który jest implementowany przez szereg klas, realizujących różne typy zadań (podział danych - `PartitionerGraphNode`, scalanie - `MergerGraphNode`, przetwarzanie - `ProcessorGraphNode`). Należy zauważyć, że opisana tu struktura nie zawiera w sobie najważniejszej części każdego z zadań - ciała metody obliczeniowej, wypełnianej przez użytkownika. Wynika to z różnic w jej zapisie po stronie Narzędzia Wspomagającego i Silnika Wykonawczego. Abstrakcyjna klasa `AbstractGraphNodeDecorator` pełni rolę klasy bazowej dla klas rozszerzających strukturę o elementy występujące tylko po jednej ze stron.

Listing 6.2: Struktura opisująca elementy scenariusza występujące w Silniku Wykonawczym

```
public class EngineGraphNodeDecorator extends AbstractGraphNodeDecorator {

    List<IKernelString> kernels;

    public List<IKernelString> getKernels() {
        return kernels;
    }

    public void setKernels(List<IKernelString> kernels) {
        this.kernels = kernels;
    }

    @Override
    public List<ValidationResult> validate() {
        (...)
    }
}

public interface IKernelString {

    String getKernel();
    Map<String, Object> getProperties();
    String getId();
    List<ValidationResult> validate();
}

public class KernelString implements IKernelString {

    private String id;
    private String kernel;
    private Map<String, Object> properties;

    (...)

    @Override
    public String getKernel() {
        return kernel;
    }

    @Override
    public Map<String, Object> getProperties() {
        return properties;
    }

    @Override
    public String getId() {
        return id;
    }

    @Override
    public List<ValidationResult> validate() {
```

```

        (...)
    }
}

```

Klasa `EngineGraphNodeDecorator` dziedziczy po `AbstractGraphNodeDecorator` i rozszerza jej działanie o agregację obiektów typu `IKernelString`, które przechowują ciała metod obliczeniowych danego węzła scenariusza w postaci stringów.

Listing 6.3: Struktura opisująca elementy scenariusza występujące w Narzędziu Wspomagającym

```

public class GUIGraphNodeDecorator extends AbstractGraphNodeDecorator {

    List<ISourceFile> sourceFiles;
    int x, y;

    (...)

    @Override
    public List<ValidationResult> validate() {
        (...)
    }

    public int getX() {
        return x;
    }

    public int getY() {
        return y;
    }

    public List<ISourceFile> getSourceFiles() {
        return sourceFiles;
    }
}

public interface ISourceFile {
    File getFile();
    Map<String, Object> getProperties();
    void setProperties(Map<String, Object> properties);
    String getId();
    List<ValidationResult> validate();
}

public class SourceFile implements ISourceFile, Serializable {

    (...)
    protected File file;
    protected String id;
    protected Map<String, Object> properties;
}

```

```

        (...)

    @Override
    public File getFile() {
        return file;
    }

    @Override
    public String getId() {
        return id;
    }

    @Override
    public Map<String, Object> getProperties() {
        return properties;
    }

    @Override
    public List<ValidationResult> validate() {
        (...)
    }

    @Override
    public void setProperties(Map<String, Object> properties) {
        this.properties = properties;
    }
}

```

Klasa `GUIGraphNodeDecorator` dziedziczy po `AbstractGraphNodeDecorator` i rozszerza jej działanie o przechowywanie informacji na temat pozycji danego węzła grafu na obszarze roboczym (zmienne `x,y`) oraz agreguje obiekty typu `ISourceFile`, które przechowują ciała metod obliczeniowych danego węzła scenariusza w postaci pliku na dysku.

Listing 6.4: Usługa walidacji scenariusza

```

public class GraphValidator {

    public static List<ValidationResult> validateGraphForGUI(
        List<GUIGraphNodeDecorator> graph) {
        List<ValidationResult> result = new ArrayList<ValidationResult>();
        List<IGraphNode> graphNodes = GraphNodeExtractor
            .extractGraphNodesForGUI(graph);
        result.addAll(validateStartGraphNodes(graphNodes));
        result.addAll(validateEndGraphNodes(graphNodes));
        result.addAll(validateChildrenGraphNodes(graphNodes));
        for (GUIGraphNodeDecorator node : graph) {
            result.addAll(node.validate());
        }
        return result;
    }
}

```

```

    public static List<ValidationResult> validateGraphForEngine(
        List<EngineGraphNodeDecorator> graph) {
        List<ValidationResult> result = new ArrayList<ValidationResult>();
        List<IGraphNode> graphNodes = GraphNodeExtractor
            .extractGraphNodesForEngine(graph);
        result.addAll(validateStartGraphNodes(graphNodes));
        result.addAll(validateEndGraphNodes(graphNodes));
        result.addAll(validateChildrenGraphNodes(graphNodes));
        for (EngineGraphNodeDecorator a : graph) {
            result.addAll(a.validate());
        }
        return result;
    }

    private static List<ValidationResult> validateStartGraphNodes(
        List<IGraphNode> graphNodes) {
        (...)
    }

    private static List<ValidationResult> validateEndGraphNodes(
        List<IGraphNode> graphNodes) {
        (...)
    }

    private static List<ValidationResult> validateChildrenGraphNodes(
        List<IGraphNode> graphNodes) {
        (...)
    }
}

```

Zaprojektowana struktura opisująca scenariusz ma duży wpływ na sposób realizacji obydwu procesów serializacji/deserializacji, które, mimo opisanych wcześniej różnic, są do siebie dość podobne. Rdzeniem ich funkcjonalności jest tutaj sposób zapisywania struktury scenariusza. Ponieważ jest to duża część działania tego podsystemu, zdecydowano się na zastosowanie mechanizmów paradygmatu programowania obiektowego, by maksymalnie zredukować ilość nadmiarowego kodu.

Listing 6.5: Struktura realizująca proces serializacji/deserializacji

```

public interface IGraphConfiguration {
    void setConfigurationFile(File file);
    File getConfigurationFile();
    String getProjectName() throws ConfigurationException;
    void setProjectName(String name) throws ConfigurationException;
    void setGraph(List<IGraphNode> nodes) throws ConfigurationException;
    void save() throws ConfigurationException;
    void save(File file) throws ConfigurationException;
    List<IGraphNode> loadGraph() throws ConfigurationException;
    List<IGraphNode> loadGraph(File file) throws ConfigurationException;
}

```

```

public interface IGUIGraphConfiguration extends IGraphConfiguration {
    List<GUIGraphNodeDecorator> loadGraphForGUI()
        throws ConfigurationException;
    List<GUIGraphNodeDecorator> loadGraphForGUI(File file)
        throws ConfigurationException;
    void saveGraphForGUI(List<GUIGraphNodeDecorator> guiGraphNodes)
        throws ConfigurationException;
    void saveGraphForGUI(List<GUIGraphNodeDecorator> guiGraphNodes, File file)
        throws ConfigurationException;
}

public interface IEngineGraphConfiguration extends IGraphConfiguration {
    List<EngineGraphNodeDecorator> loadGraphForEngine()
        throws ConfigurationException;
    List<EngineGraphNodeDecorator> loadGraphForEngine(File file)
        throws ConfigurationException;
    void saveGraphForEngine(List<EngineGraphNodeDecorator> graphNodes)
        throws ConfigurationException;
    void saveGraphForEngine(List<EngineGraphNodeDecorator> graphNodes, File file)
        throws ConfigurationException;
    String getInputDataURL() throws ConfigurationException;
    void setInputDataURL(String inputDataURL) throws ConfigurationException;
    void saveGraphForEngine(List<EngineGraphNodeDecorator> graphNodes, Writer writer)
        throws ConfigurationException;
    List<EngineGraphNodeDecorator> loadGraphForEngine(Reader reader)
        throws ConfigurationException;
}

public abstract class AbstractGraphConfiguration implements IGraphConfiguration {
    (...)
    protected XMLConfiguration config;
    (...)
}

public class GUIGraphConfiguration extends AbstractGraphConfiguration
    implements IGUIGraphConfiguration {
    (...)
}

public class EngineGraphConfiguration extends AbstractGraphConfiguration
    implements IEngineGraphConfiguration {
    (...)
}

```

Bazowym interfejsem tego podsystemu jest `IGraphConfiguration`. Definiuje on operacje na elementach występujących i po stronie Narzędzia Wspomagającego, i Silnika Wykonawczego. Interfejsy `IEngineGraphConfiguration` i `IGUIGraphConfiguration` dziedziczące po nim opisują metody działające na częściach składowych scenariusza występujących tylko po jednej ze stron. Klasy realizujące te interfejsy, `GUIGraphConfiguration` i `EngineGraphConfiguration` dziedziczą po klasie

AbstractGraphConfiguration, implementującej interfejs IGraphConfiguration. Serce działania całego podsystemu jest obiekt typu XMLConfiguration, klasy będącej częścią biblioteki Apache Commons Configuration¹, udostępniającej metody odczytu i zapisu dla różnych formatów konfiguracji programu. W tym wypadku zdecydowano się na zapis w formacie XML.

Listing 6.6: Przykład zserializowanego scenariusza

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<kh:project name="hive-project">
  <kh:node hash="130353709" id="e038c1db-c4d9-4a61-b188-495c81200125" name="name1"
    parent-id="" type="DataPartitioner" x="200" y="130">
    <kh:first-children-nodes/>
    <kh:send-to>
      <kh:following-node id="8b584317-997f-4ef6-998a-86a1f21ff60b"/>
      <kh:following-node id="cbb7d846-ec04-4b8a-a9f6-a85bac9c694f"/>
    </kh:send-to>
    <kh:properties/>
    <kh:node-source-files>
      <kh:source-file id="eff1aa1a-686b-49bb-8ba5-47f607f27a45" src="./
        e038c1db-c4d9-4a61-b188-495c81200125/kernel.cl">
        <kh:property key="offsets" value="0"/>
        <kh:property key="globalSizes" value="1"/>
        <kh:property key="numberOfDimensions" value="1"/>
        <kh:property key="localSizes" value="1"/>
      </kh:source-file>
    </kh:node-source-files>
  </kh:node>
  <kh:node hash="-431625787" id="a296c544-84cf-4886-a786-d13ae2a580a8" name="name2"
    parent-id="" type="DataMerger" x="189" y="290">
    <kh:first-children-nodes/>
    <kh:send-to/>
    <kh:properties/>
    <kh:node-source-files>
      <kh:source-file id="680d7b83-c327-4d73-887f-3d4ed3ae14c4" src="./
        a296c544-84cf-4886-a786-d13ae2a580a8/kernel.cl">
        <kh:property key="offsets" value="0"/>
        <kh:property key="globalSizes" value="1"/>
        <kh:property key="numberOfDimensions" value="1"/>
        <kh:property key="localSizes" value="1"/>
      </kh:source-file>
    </kh:node-source-files>
  </kh:node>
  <kh:node hash="1905847244" id="8b584317-997f-4ef6-998a-86a1f21ff60b" name="name3"
    parent-id="" type="DataProcessor" x="120" y="210">
    <kh:first-children-nodes/>
    <kh:send-to>
      <kh:following-node id="a296c544-84cf-4886-a786-d13ae2a580a8"/>
    </kh:send-to>
    <kh:properties/>
    <kh:node-source-files>
      <kh:source-file id="bc0514aa-1716-46b1-8d5f-5a33dbf9f677" src="
```

¹<http://commons.apache.org/configuration/>

```

        ./8b584317-997f-4ef6-998a-86a1f21ff60b/kernel.cl">
        <kh:property key="offsets" value="0"/>
        <kh:property key="globalSizes" value="1"/>
        <kh:property key="numberOfDimensions" value="1"/>
        <kh:property key="localSizes" value="1"/>
    </kh:source-file>
</kh:node-source-files>
</kh:node>
<kh:node hash="-508184550" id="cbb7d846-ec04-4b8a-a9f6-a85bac9c694f" name="name5"
parent-id="" type="DataProcessor" x="258" y="210">
    <kh:first-children-nodes/>
    <kh:send-to>
        <kh:following-node id="a296c544-84cf-4886-a786-d13ae2a580a8"/>
    </kh:send-to>
    <kh:properties/>
    <kh:node-source-files>
        <kh:source-file id="bc0514aa-1716-46b1-8d5f-5a33dbf9f677" src="./
cbb7d846-ec04-4b8a-a9f6-a85bac9c694f/kernel.cl">
            <kh:property key="offsets" value="0"/>
            <kh:property key="globalSizes" value="1"/>
            <kh:property key="numberOfDimensions" value="1"/>
            <kh:property key="localSizes" value="1"/>
        </kh:source-file>
    </kh:node-source-files>
</kh:node>
</kh:project>

```

6.5.2 Komunikacja z Silnikiem Wykonawczym

W celu poprawnego rozwiązania problemu komunikacji z Silnikiem Wykonawczym zdecydowano się na wykorzystanie w programie architektury wielowątkowej. W takim podejściu aplikacja przetwarza zadania użytkownika w wątku głównym, natomiast wszelkie operacje wymagające większej ilości czasu (w tym i komunikacja sieciowa) są wykonywane w osobnym wątku. Dzięki temu użytkownik nie doświadcza nieproduktywnych przestojów w działaniu aplikacji, kiedy ta realizuje długotrwałe operacje. Przykładowo, gdy użytkownik wysyła scenariusz do obliczenia nie musi czekać, aż wszystkie dane zostaną wysłane i odebrane przez Silnik Wykonawczy - po zainicjowaniu tej akcji może zająć się innymi zadaniami (np. utworzyć nowy scenariusz), a przesył danych będzie odbywać się w tle.

Standaryzacja sposobu komunikacji z silnikiem Wykonawczym była jednym z nieformalnych wymogów nałożonych na tą funkcję. Po zakończeniu prac zamierzano udostępnić kod źródłowy całego systemu pod licencją GPL, by umożliwić społeczności Open Source swobodne dodawanie nowych elementów systemu - na przykład aplikację mobilną wyświetlającą postęp wykonania scenariusza. Zdecydowano się zatem na wykorzystanie technologii usług sieciowych opartej na protokole SOAP.

Takie podejście ma wiele zalet - komunikacja odbywa się w znormalizowany sposób, nie jest ograniczona do jednego medium warstwy transportowej modelu OSI (można używać protokołu HTTP, JMS, SMTP, komunikacji opartej na socketach itd.), dzięki domyślnemu użyciu protokołu HTTP nieistotny jest sposób konfiguracji połączenia sieciowego klienta (firewalle, serwery proxy, NAT) oraz jest niezależna od wybranego modelu programistycznego.

Silnik Wykonawczy udostępnia więcej niż jedną usługę sieciową, którą mógłby wywołać użytkownik. Przy założeniu istnienia dwóch znaczących wątków w programie - głównego i komunikacyjnego, wywołanie więcej niż jednej usługi (np wysłanie scenariusza do wykonania i wyświetlenie aktualnego obciążenia systemu) spowodowałoby długi czas oczekiwania na odpowiedź. Rozwiązaniem naiwnym tego problemu byłoby uruchamianie każdego wywołania usługi w osobnym wątku. Nie jest to jednak słuszne podejście - pomijając aspekt wydajnościowy (tworzenie za każdym razem nowego wątku jest kosztowne), pozwolenie użytkownikowi na wykonywanie nieograniczonej liczby wywołań stanowi poważną lukę w bezpieczeństwie krytycznej części systemu, jaką jest Silnik Wykonawczy. Biorąc powyższe aspekty pod uwagę, zdecydowano się na utworzenie puli wątków komunikacyjnych. W ten sposób użytkownik może, w ramach normalnej pracy z aplikacją, wywoływać potrzebne mu usługi sieciowe, natomiast górne ograniczenie liczby wątków w puli chroni system przez niewiedzą lub złą wolą użytkownika.

Listing 6.7: Usługa sieciowa Silnika Wykonawczego

```
public interface ClientBean {  
    @WebMethod  
    public String getWorkflowResults(Integer arg0);  
  
    @WebMethod  
    public List<ClusterInfo> browseInfrastructure();  
  
    @WebMethod  
    public Integer submitWorkflow(String arg0);  
  
    @WebMethod  
    public List<WorkflowInfo> browseWorkflows();  
  
    @WebMethod  
    public void terminateWorkflow(Integer arg0);  
}
```

Powyższy listing prezentuje interfejs usługi sieciowej oferowanej przez Silnik Wykonawczy. Ponieważ są to metody blokujące, ich wywołanie należało przenieść z wątku głównego aplikacji do osobnego wątku komunikacyjnego.

Listing 6.8: Realizacja usługi udostępniającej pulę wątków komunikacyjnych

```

public interface IExecutionEngineService {
    void browseInfrastructure(ExecutionEngineServiceListener listener);

    void getWorkflowResults(Integer workflowId,
        ExecutionEngineServiceListener listener);

    void terminateWorkflow(Integer workflowId,
        ExecutionEngineServiceListener listener);

    void browseWorkflows(ExecutionEngineServiceListener listener);

    void submitWorkflow(String workflowStream,
        ExecutionEngineServiceListener listener);
}

public class ExecutionEngineService implements IExecutionEngineService {

    private ExecutorService executorService;
    private ClientBeanService clientService;

    public ExecutionEngineService() throws ExecutionEngineServiceException{
        executorService = Executors.newFixedThreadPool(5);
        try{
            clientService = new ClientBeanService();
        } catch(WebServiceException e){
            throw new ExecutionEngineServiceException(e);
        }
    }

    @Override
    public void browseInfrastructure(final ExecutionEngineServiceListener listener) {
        Runnable r = new Runnable() {

            @Override
            public void run() {
                List<ClusterInfo> clusterInfo = clientService.
                    getClientBeanPort().browseInfrastructure();
                listener.infrastructureBrowseCompleted(clusterInfo);
            }
        };
        executorService.execute(r);
    }

    (...)
}

public interface ExecutionEngineServiceListener {
    void infrastructureBrowseCompleted(List<ClusterInfo> clusterInfo);
    void getWorkflowResultsCompleted(String result);
    void submitWorkflowCompleted(Integer workflowId);
    void workflowBrowseCompleted(List<WorkflowInfo> workflowInfo);
    void terminateWorkflowCompleted();
}

```

```
public abstract class ExecutionEngineServiceListenerAdapter
    implements ExecutionEngineServiceListener {

    @Override
    public void infrastructureBrowseCompleted(List<ClusterInfo> clusterInfo) { }

    @Override
    public void getWorkflowResultsCompleted(String result) { }

    @Override
    public void submitWorkflowCompleted(Integer workflowId) { }

    @Override
    public void workflowBrowseCompleted(List<WorkflowInfo> workflowInfo) { }

    @Override
    public void terminateWorkflowCompleted() { }
}
```

Klasa `ExecutionEngineService` pełni rolę fasady dla usługi sieciowej oferowanej przez Silnik Wykonawczy. Do utworzenia i utrzymywania puli wątków komunikacyjnych wykorzystywany jest obiekt o interfejsie `ExecutorService`², oferujący możliwość uruchamiania obiektów typu `Runnable`³ jako osobne wątki w ramach zdefiniowanej puli. Przy jako parametr metod klasy `ExecutionEngineService` podawany jest obiekt o interfejsie `ExecutionEngineServiceListener`, do którego następuje odwołanie po zakończeniu działania metody usługi sieciowej, w ramach mechanizmu wywołania zwrotnego (ang. *callback*). By ułatwić wykorzystanie tej struktury programistycznej, wprowadzono abstrakcyjną klasę `ExecutionEngineServiceListenerAdapter` implementującą ten interfejs. Dzięki temu programista może, dziedzicząc po niej, zaimplementować tylko te metody, których aktualnie potrzebuje.

Scenariusze wysyłane do Silnika Wykonawczego nierzadko są bardzo rozbudowane - kilkadziesiąt lub kilkaset węzłów ze skomplikowanymi metodami obliczeniowymi. Należało upewnić się, że przesyłane dane są dostarczane w całości w swej niezmienionej formie. Oprócz weryfikacji spójności transmitowanych danych, koniecznym również było upewnienie się, że wysyłany scenariusz spełnia warunki, wymagane do jego wykonania. Powyższe cele osiągnięto poprzez wprowadzenie dwustopniowej walidacji. Użytkownik inicjując akcję wysłania scenariusza do wykonania uruchamia jednocześnie pierwszy stopień walidacji. Polega on na sprawdzeniu, czy konstrukcja scenariusza spełnia założenia i warunki, które określają go jako poprawnie skonstruowany. Następnie następuje serializacja scenariusza do formy zdatnej do przesyłu do Silnika Wykonawczego. Podczas

²<http://docs.oracle.com/javase/6/docs/api/java/util/concurrent/ExecutorService.html>

³<http://docs.oracle.com/javase/6/docs/api/java/lang/Runnable.html>

serializacji dokonywane jest obliczenie i zapisanie sumy kontrolnej każdego z zadań obliczeniowych scenariusza. Drugi etap walidacji wykonywany jest po stronie Silnika Wykonawczego, po otrzymaniu scenariusza do wykonania. W trakcie deserializacji scenariusza ponownie obliczana jest suma kontrolna każdego z zadań obliczeniowych, a następnie porównywana z poprzednio wyliczoną. Jeżeli wszystkie sumy się zgadzają, możemy być pewni, że scenariusz został przesłany w niezmienionej formie.

Listing 6.9: Struktura przechowująca wynik walidacji

```
public class ValidationResult implements Serializable {

    (...)

    public static enum ValidationResultType{
        VALID, INVALID
    };

    private final String message;
    private final ValidationResultType type;

    (...)

}
```

Listing 6.10: Klasa odpowiedzialna za walidację scenariusza

```
public class GraphValidator {

    public static List<ValidationResult> validateGraphForGUI(
        List<GUIGraphNodeDecorator> graph) {
        List<ValidationResult> result = new ArrayList<ValidationResult>();
        List<IGraphNode> graphNodes = GraphNodeExtractor
            .extractGraphNodesForGUI(graph);
        result.addAll(validateGraphNodes(graphNodes));
        for (GUIGraphNodeDecorator node : graph) {
            result.addAll(node.validate());
        }
        return result;
    }

    public static List<ValidationResult> validateGraphForEngine(
        List<EngineGraphNodeDecorator> graph){
        List<ValidationResult> result = new ArrayList<ValidationResult>();
        List<IGraphNode> graphNodes = GraphNodeExtractor
            .extractGraphNodesForEngine(graph);
        result.addAll(validateGraphNodes(graphNodes));
        for (EngineGraphNodeDecorator a : graph) {
            result.addAll(a.validate());
        }
        return result;
    }

    private static List<ValidationResult> validateGraphNodes(
```

```
        List<IGraphNode> graphNodes) {
    List<ValidationResult> results = new ArrayList<ValidationResult>();
    results.addAll(validateStartGraphNodes(graphNodes));
    results.addAll(validateEndGraphNodes(graphNodes));
    results.addAll(validateChildrenGraphNodes(graphNodes));
    for (IGraphNode node : graphNodes) {
        results.addAll(node.validate());
    }
    return results;
}

private static List<ValidationResult> validateStartGraphNodes(
    List<IGraphNode> graphNodes) {
    (...)
}

private static List<ValidationResult> validateEndGraphNodes(
    List<IGraphNode> graphNodes) {
    (...)
}

private static List<ValidationResult> validateChildrenGraphNodes(
    List<IGraphNode> graphNodes) {
    (...)
}
}
```

6.5.3 Komunikacja z Repozytorium Szablonów Obliczeniowych

Mając na uwadze wymagania i potrzeby zidentyfikowane w trakcie projektowania systemu, zdecydowano się na fizyczne rozdzielenie kodu Narzędzia Wspomagającego i kodu Repozytorium Szablonów Obliczeniowych. Zrealizowano to poprzez wydzielenie osobnego projektu z klasami, które wykorzystywane są i w Narzędziu Wspomagającym, i w Silniku Wykonawczym i umieszczeniu go logicznie w ramach komponentu hive-common. Projekt ten budowany jest do postaci archiwum JAR, które następnie jest ładowane i wykorzystywane jest przez obydwie wymienione elementy systemu. Dzięki takiemu rozwiązaniu nie ma potrzeby przebudowywania aplikacji za każdym razem, kiedy dodany zostanie nowy typ zadania obliczeniowego.

Wymaganie uwspólnienia wersji powyższego archiwum JAR dla Narzędzia Wspomagającego i Silnika Wykonawczego wymagało dłuższego zastanowienia. Problem ten rozwiązano, wykorzystując możliwość ładowania archiwów JAR poprzez ich adresy URL. W ten sposób, umieszczając archiwum JAR w miejscu widocznym i dla Narzędzia Wspomagającego, i dla Silnika Wykonawczego zyskano pewność, że w każdym momencie czasu obydwie elementy systemu będą używały tej samej wersji biblioteki.

6.5.4 Edycja kodu źródłowego metod obliczeniowych

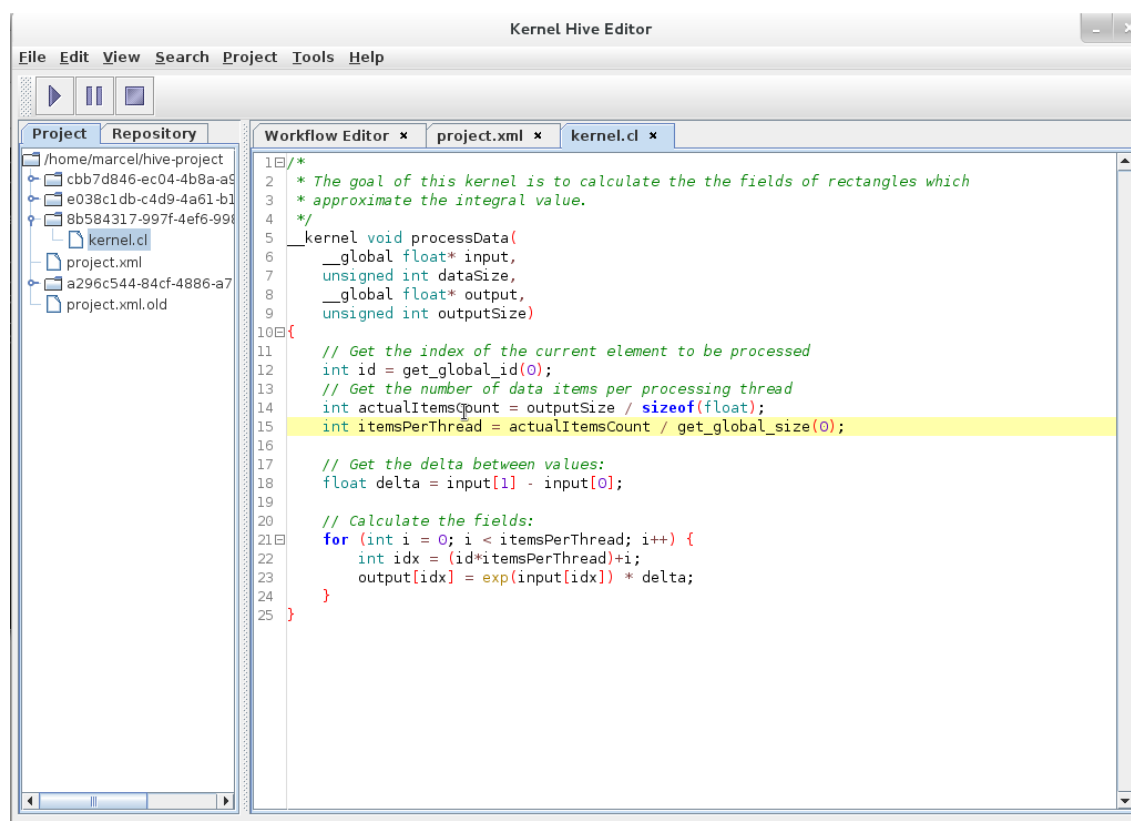
Jedną z podstawowych funkcji Narzędzia Wspomagającego jest możliwość edycji ciał metod obliczeniowych, dostarczanych za pomocą szablonów zadań scenariusza. Ponieważ jednym z założeń systemu „Kernel Hive” było umożliwienie użytkownikowi wykorzystania procesorów CPU i GPU, zdecydowano, że metody obliczeniowe będą zapisane w języku OpenCL⁴, opartym na specyfikacji języka C - C99. Podczas implementacji możliwości edycji ciał metod skupiono się przede wszystkim na łatwości użycia, funkcjonalności i przejrzystości edytora. Ponieważ aplikacja wykorzystuje bibliotekę Java Swing, zdecydowano się na użycie takiego jej komponentu, który spełniałby powyższe założenia. Po przejrzeniu dostępnych rozwiązań zdecydowano się użyć komponentu RSyntaxTextArea⁵. Oto niektóre z jego cech świadczących o użyteczności:

- Podświetlanie składni ponad 30 języków programowania
- Możliwość zwijania wierszy
- Wyszukiwanie/zamiana fraz ze standardowymi funkcjami (dopasowanie wielkości liter, wyrażenia regularne)
- Wczytywanie i zapisywanie plików lokalnych i zdalnych (za pomocą protokołu FTP)
- Możliwość definicji własnych makr
- Funkcjonalność cofnij/powtórz (ang. *undo/redo*)
- Drag-and-drop - przeciąganie tekstu z i do edytora
- Dopasowywanie nawiasów
- Podświetlanie linii z kursorem

Dzięki powyższym cechom, komponent RSyntaxTextArea oferuje funkcjonalność niezbyt odbiegającą od nowoczesnych edytorów wchodzących w skład wiodących zintegrowanych środowisk deweloperskich (IDE). Komponent jest dostępny pod zmodyfikowaną licencją BSD.

⁴<http://www.khronos.org/opencv/>

⁵<http://fifesoft.com/rsyntaxtextarea/>



Rysunek 6.1: Przykład działania edytora kodu

6.5.5 Sposób tworzenia scenariuszy obliczeniowych

Poprawne zaprojektowanie i zaimplementowanie łatwego i intuicyjnego sposobu tworzenia scenariuszy było jednym z kluczowych aspektów działania Narzędzia Wspomagającego. Mając na uwadze wymagania i potrzeby zidentyfikowane podczas projektowania systemu, zdecydowano się na umożliwienie użytkownikowi tworzenia scenariuszy w sposób graficzny. Projektowanie scenariusza polega tym samym na wizualnej edycji grafu reprezentującego projektowany scenariusz.

Innymi słowy, użytkownik (za pomocą myszki i mechanizmu drag-and-drop) przeciąga szablony zadań obliczeniowych z Repozytorium Szablonów Obliczeniowych na zdefiniowaną przestrzeń roboczą okna aplikacji, kopiując je, gdzie pozostają widoczne jako węzły grafu. Ma również możliwość edycji poprzednio dodanych węzłów (zmiana ich położenia na przestrzeni roboczej, zmiana ich właściwości) oraz łączenia ich krawędziami, które reprezentują zależności czasowe między zadaniami obliczeniowymi.

Wybór odpowiedniej kontrolki umożliwiającej powyższe operacje wymagał wzięcia pod uwagę dodatkowych wymagań, wynikających z ustalonej charakterystyki scenariuszy wykonywanych w

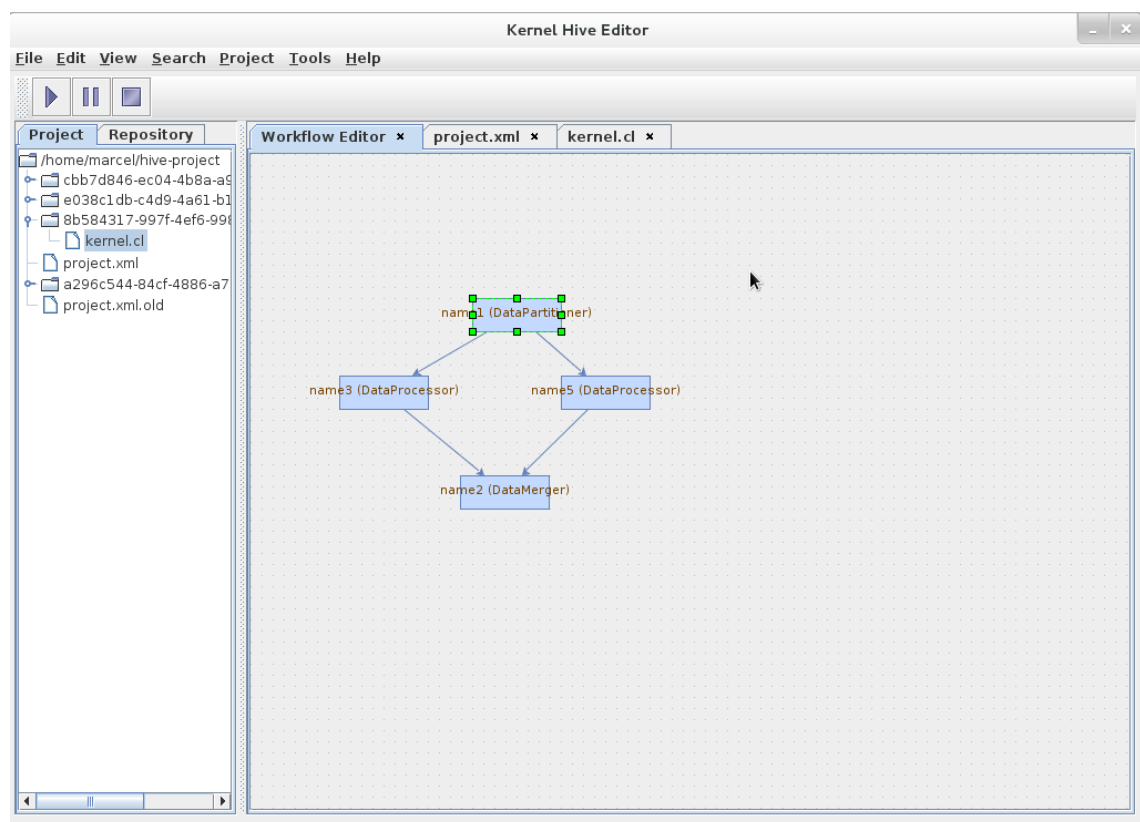
systemie „Kernel Hive”. W tym wypadku główną charakterystyką sterującą wyborem była możliwość dodawania dzieci do danego węzła. Jest to cecha nieobecna w części bibliotek i narzędzi rysujących grafy, co zawężyło pole wyboru.

Jednocześnie, ponieważ wykorzystano bibliotekę graficzną Java Swing do zaprojektowania wyglądu aplikacji, w trakcie poszukiwań preferowano rozwiązania, które opierały się na tej bibliotece. Wybór ten dokonano ze względów praktycznych - starano się maksymalnie uniezależnić od dodatkowych bibliotek i liczone się z faktem, że wybrana kontrolka może zostać poddana modyfikacjom, by lepiej dopasować się do wymagań.

Mając powyższe rozważania na uwadze, zdecydowano się wykorzystać do tego celu bibliotekę JGraphX⁶, wydaną pod licencją BSD. Spełnia wszystkie wymagania postawione w fazie projektowania. Oto niektóre z jej możliwości:

- Umożliwia tworzenie hierarchicznych grafów
- Konfigurowalny wygląd wierzchołków i krawędzi
- W pełni zintegrowany z biblioteką Java Swing
- Funkcjonalność cofnij/powtórz (*undo/redo*)
- Drag-and-drop wierzchołków na przestrzeń roboczą
- Automatyczne zastosowanie jednego z predefiniowanych układów wierzchołków grafu

⁶<http://www.jgraph.com/jgraph.html>



Rysunek 6.2: Przykład działania edytora scenariusza

Rozdział 7

Realizacja i ocena systemu

7.1 Krótkie streszczenie rozdziału

Ostatnim etapem w cyklu wytworzenia usługi było testowanie działania programu oraz ewaluacja otrzymanych wyników. Testy akceptacyjne zostały wykonane przez grupę testerów, a otrzymane dane zostały poddane analizie. W trakcie testu skupiono się na łatwości użytkowania i intuicyjności aplikacji, by następnie, na podstawie uzyskanych informacji dokonać oceny tych parametrów.

W tym rozdziale opisane zostały warunki testowe, otrzymane wyniki, ich analiza oraz proponowane możliwości rozbudowy systemu o nowe funkcjonalności.

7.2 Docelowa architektura i opis komponentów

7.2.1 Docelowa architektura aplikacji

System został zrealizowany w postaci aplikacji desktopowej, działającej lokalnie na komputerze użytkownika. Można w nim wyróżnić 12 podsystemów.

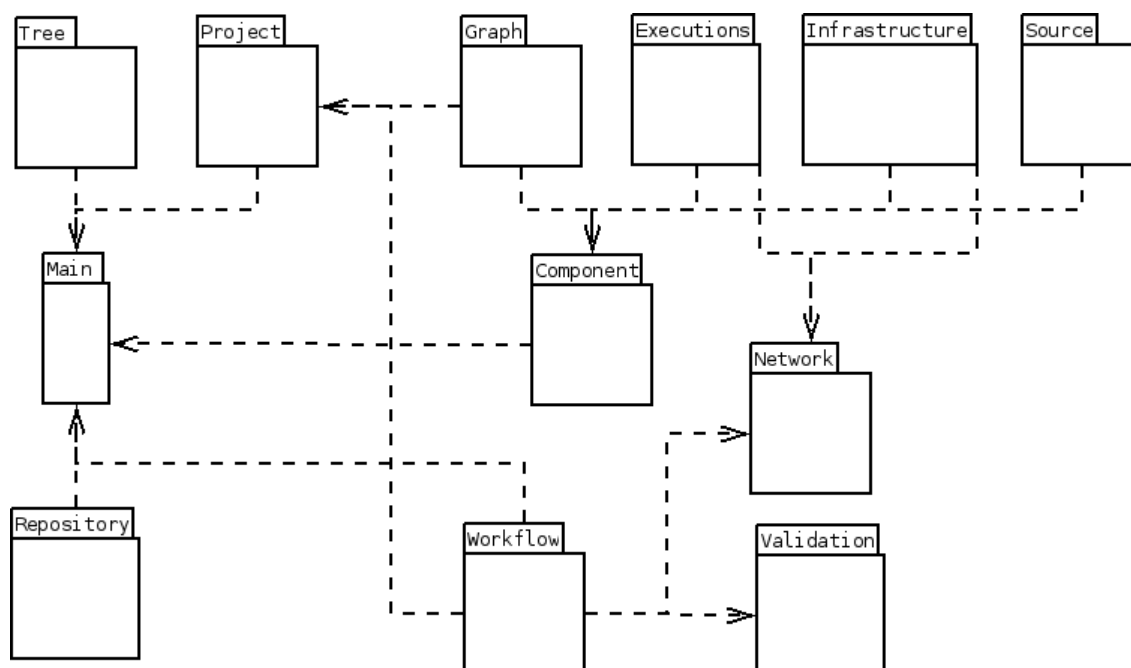
- Podsystem główny odpowiedzialny za konfigurację, uruchomienie aplikacji oraz wyświetlanie bazowych komponentów graficznych (main)
- Podsystem odpowiedzialny za operacje manipulujące projektem scenariusza obliczeniowego (project)
- Podsystem odpowiedzialny za wyświetlanie szablonów obliczeniowych (repository)

- Podsystem odpowiedzialny za operacje wykonywane podczas wysyłania scenariusza obliczeniowego do wykonania (workflow)
- Podsystem odpowiedzialny za realizację komunikacji z innymi komponentami systemu „Kernel Hive” (network)
- Podsystem odpowiedzialny za operacje wykonywane na grafie - graficznej reprezentacji scenariusza (graph)
- Podsystem odpowiedzialny za walidację scenariusza obliczeniowego (validation)
- Podsystem odpowiedzialny za wyświetlanie danych na temat wykonania scenariuszy obliczeniowych użytkownika (executions)
- Podsystem odpowiedzialny za wyświetlanie danych na temat obecnej konfiguracji infrastruktury sprzętowej systemu ”Kernel Hive” (infrastructure)
- Podsystem odpowiedzialny za operacje wykonywane na pliku z kernelem obliczeniowym zadania scenariusza (source)
- Podsystem odpowiedzialny za wyświetlanie danych na temat struktury katalogowej projektu scenariusza (tree)
- Podsystem pozwalający na łatwe otwieranie i zarządzanie kartami w aplikacji (component)

Poszczególne podsystemy są od siebie zależne w sposób przedstawiony na diagramie 7.1.

7.2.2 Opis podsystemów

SSYS_001	main
Opis:	Podsystem main odpowiada za uruchomienie systemu, obsługę sygnałów systemowych, ładowanie zmiennych konfiguracyjnych oraz za obsługę głównego okna aplikacji i akcji z nim związanych
Przypadki użycia:	Uruchomienie aplikacji, zamknięcie aplikacji
Lokalizacja:	HCMP_001 Stacja robocza użytkownika
Komponenty:	Plik główny programu (KernelHiveMain.java), klasa głównego okna (MainFrame), klasa ze zmiennymi konfiguracyjnymi aplikacji (AppConfiguration.java)
Powiązania:	Podsystemy: tree, workflow, component, repository



Rysunek 7.1: Diagram interakcji podsystemów aplikacji

SSYS_002	project
Opis:	Podsystem project odpowiada za zapisywanie i przechowywanie danych projektu scenariusza obliczeniowego
Przypadki użycia:	Otwarcie projektu scenariusza, utworzenie nowego projektu scenariusza
Lokalizacja:	HCMP_001 Stacja robocza użytkownika
Komponenty:	IProject.java, KernelHiveProject.java
Powiązania:	Podsystem main

SSYS_003	repository
Opis:	Podsystem repository odpowiada za wyświetlanie i pobieranie szablonów obliczeniowych zdefiniowanych w repozytorium szablonów obliczeniowych
Przypadki użycia:	Pobranie nowego węzła obliczeniowego celem dodania go do scenariusza
Lokalizacja:	HCMP_001 Stacja robocza użytkownika
Komponenty:	Plik z klasą RepositoryViewer
Powiązania:	Podsystem main

SSYS_004	workflow
Opis:	Podsystem workflow odpowiada za operacje wykonywane przed i podczas wysyłania scenariusza obliczeniowego do wykonania
Przypadki użycia:	Konwersja scenariusza do formatu akceptowalnego przez Silnik Wykonawczy, Pakowanie danych niezbędnych do wykonania scenariusza, Inicjacja procesu wysłania scenariusza do wykonania
Lokalizacja:	HCMP_001 Stacja robocza użytkownika
Komponenty:	Pliki w paczce pl.gda.pg.eti.kernelhive.gui.workflow
Powiązania:	Podsystemy main, project, validation, network

SSYS_005	network
Opis:	Podsystem network odpowiada za realizację komunikacji między aplikacją a Silnikiem Wykonawczym
Przypadki użycia:	Wysłanie scenariusza do wykonania, sprawdzenie statusu wykonania scenariuszy danego użytkownika, sprawdzenie obciążenia systemu, pobranie informacji na temat konfiguracji systemu
Lokalizacja:	HCMP_001 Stacja robocza użytkownika
Komponenty:	WorkflowService, InfrastructureService, MonitorService
Powiązania:	Podsystemy workflow, executions, infrastructure

SSYS_06	graph
Opis:	Podsystem graph odpowiada za operacje wykonywane na grafie, będącym reprezentacją scenariusza obliczeniowego
Przypadki użycia:	Dodanie nowego węzła do grafu, połączenie węzłów grafu krawędzią, usunięcie węzła z grafu
Lokalizacja:	HCMP_001 Stacja robocza użytkownika
Komponenty:	GraphNode, GraphNodeType, GraphConfiguration
Powiązania:	Podsystemy component, project

SSYS_07	validation
Opis:	Podsystem validation odpowiada za walidację scenariusza obliczeniowego i wykrywania ewentualnych nieprawidłowości lub błędów w jego konstrukcji
Przypadki użycia:	Sprawdzenie poprawności scenariusza obliczeniowego przed wysłaniem go do wykonania
Lokalizacja:	HCMP_001 Stacja robocza użytkownika
Komponenty:	Pliki w paczce pl.gda.pg.eti.kernelhive.common.validation
Powiązania:	Podsystem workflow

SSYS_08	executions
Opis:	Podsystem executions odpowiada za wyświetlenie danych na temat wykonania scenariuszy obliczeniowych użytkownika
Przypadki użycia:	Sprawdzenie wykonania scenariuszy obliczeniowych, Sprawdzenie wyniku wykonania scenariusza obliczeniowego
Lokalizacja:	HCMP_001 Stacja robocza użytkownika
Komponenty:	Pliki w paczce pl.gda.pg.eti.kernelhive.gui.component.workflow.viewer
Powiązania:	Podsystemy component, network

SSYS_09	infrastructure
Opis:	Podsystem infrastructure odpowiada za wyświetlenie danych na temat aktualnej konfiguracji infrastruktury sprzętowej systemu "Kernel Hive"
Przypadki użycia:	Sprawdzenie aktualnej konfiguracji infrastruktury sprzętowej systemu
Lokalizacja:	HCMP_001 Stacja robocza użytkownika
Komponenty:	Pliki w paczce pl.gda.pg.eti.kernelhive.gui.component.infrastructure
Powiązania:	Podsystemy component, network

SSYS_10	source
Opis:	Podsystem source odpowiada za operacje wykonywane na pliku z kernelem obliczeniowym zadania scenariusza
Przypadki użycia:	Edycja pliku z kernelem dostarczonym razem z szablonem obliczeniowym
Lokalizacja:	HCMP_001 Stacja robocza użytkownika
Komponenty:	Pliki w paczce pl.gda.pg.eti.kernelhive.gui.component.source
Powiązania:	Podsystem component

SSYS_11	tree
Opis:	Podsystem tree odpowiada za wyświetlanie danych na temat struktury katalogowej projektu scenariusza
Przypadki użycia:	Wyświetlenie struktury katalogowej projektu scenariusza obliczeniowego
Lokalizacja:	HCMP_001 Stacja robocza użytkownika
Komponenty:	Pliki w paczce pl.gda.pg.eti.kernelhive.gui.component.tree
Powiązania:	Podsystem component

SSYS_012	component
Opis:	Podsystem component pozwalający na łatwe otwieranie i zarządzanie kartami w aplikacji, zawierającymi w sobie główne komponenty wizualne
Przypadki użycia:	Edycja kodu źródłowego metody obliczeniowej danego węzła, Zaprezentowanie obecnej architektury systemu, Zaprezentowanie aktualnego obciążenie systemu, Graficzna edycja scenariusza obliczeniowego
Lokalizacja:	HCMP_001 Stacja robocza użytkownika
Komponenty:	Pliki w paczce pl.gda.pg.eti.kernelhive.gui.component (JTabContent.java, JTabPanel.java)
Powiązania:	Podsystemy main, graph, executions, infrastructure, source

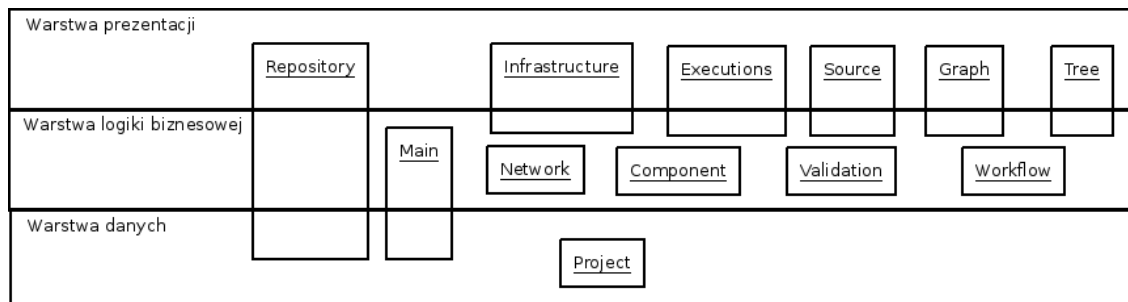
7.2.3 Opis warstw architektonicznych

Biorąc pod uwagę odpowiedzialności i sposób zachowania poszczególnych podsystemów, w aplikacji można wyróżnić 3 główne warstwy architektoniczne:

- Warstwa prezentacji, odpowiadająca za komunikację z użytkownikiem za pomocą informacji wyświetlanych w oknie aplikacji
- Warstwa logiki biznesowej, będącą główną warstwą, w której odbywa się realizacja całej logiki biznesowej sterującej wykonaniem programu
- Warstwa danych, odpowiedzialna za zapisywanie i odczytywanie projektów do/z plików, odczyt/zapis zmiennych konfiguracyjnych, odczyt szablonów obliczeniowych z repozytorium szablonów

Każdy z wyżej wymienionych podsystemów znajduje się w (co najmniej) jednej z warstw architektonicznych zdefiniowanych powyżej. Diagram 7.2 przedstawia podział podsystemów na warstwy

architektoniczne.



Rysunek 7.2: Diagram warstw architektonicznych aplikacji

7.2.4 Opis komponentów programowych

W systemie zostały wykorzystane również zewnętrzne komponenty programowe. Ich użycie miało na celu zmniejszenie czasu wytwarzania aplikacji oraz realizację jej funkcjonalności za pomocą przetestowanych i sprawdzonych rozwiązań.

SCMP_001	Biblioteka Swing
Opis:	Biblioteka graficzna wykorzystana do zaprojektowania okien aplikacji
SCMP_002	Biblioteka Apache Commons Configuration
Opis:	biblioteka odpowiedzialna za odczyt/zapis z/do plików konfiguracyjnych i plików projektów
SCMP_003	Biblioteka RSyntaxTextArea
Opis:	Podświetlający składnię komponent tekstowy zrealizowany jako rozszerzenie biblioteki Java Swing
SCMP_004	Biblioteka JGraphX
Opis:	Komponent graficzny odpowiedzialna za wizualizację grafów rozszerzający bibliotekę Java Swing

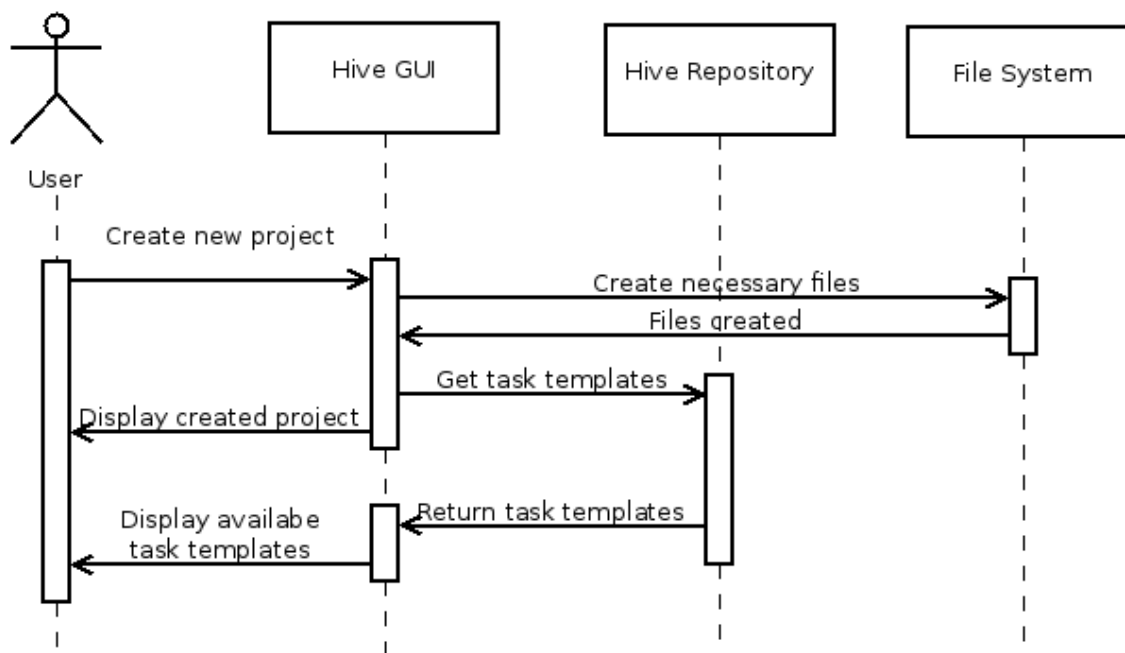
7.2.5 Główne przykłady interakcji aplikacji z systemem „Kernel Hive”

Narzędzie Wspomagające zostało zaprojektowane i zrealizowane jako część systemu „Kernel Hive”, przez co jego interakcja z innymi elementami tego systemu jest wyraźnie widoczna w przypadku realizacji każdej z głównych funkcji programowych. W tej sekcji skupiono się na opisanu wyżej wymienionych interakcji.

Utworzenie nowego projektu scenariusza

Zdarzenie to inicjuje Użytkownik, wydając polecenie utworzenia nowego projektu scenariusza obliczeniowego. Sposób działania systemu można tu podzielić na następujące fazy:

1. Aplikacja definiuje potrzebną strukturę katalogową i wysyła do lokalnego systemu plików polecenie jej utworzenia
2. System plików zwraca informacje na temat nowo utworzonych plików i katalogów
3. Aplikacja wyświetla na ekranie stosowny komunikat oraz prezentuje użytkownikowi wizualizację stworzonego katalogu projektowego
4. Aplikacja komunikuje się z komponentem Repozytorium Szablonów Obliczeniowych, by uzyskać listę szablonów dostępnych do wykorzystania w scenariuszu przez Użytkownika
5. Repozytorium Szablonów Obliczeniowych zwraca listę szablonów dostępnych do wykorzystania
6. Aplikacja ładuje widok listy szablonów obliczeniowych dostępnych do wykorzystania przez Użytkownika

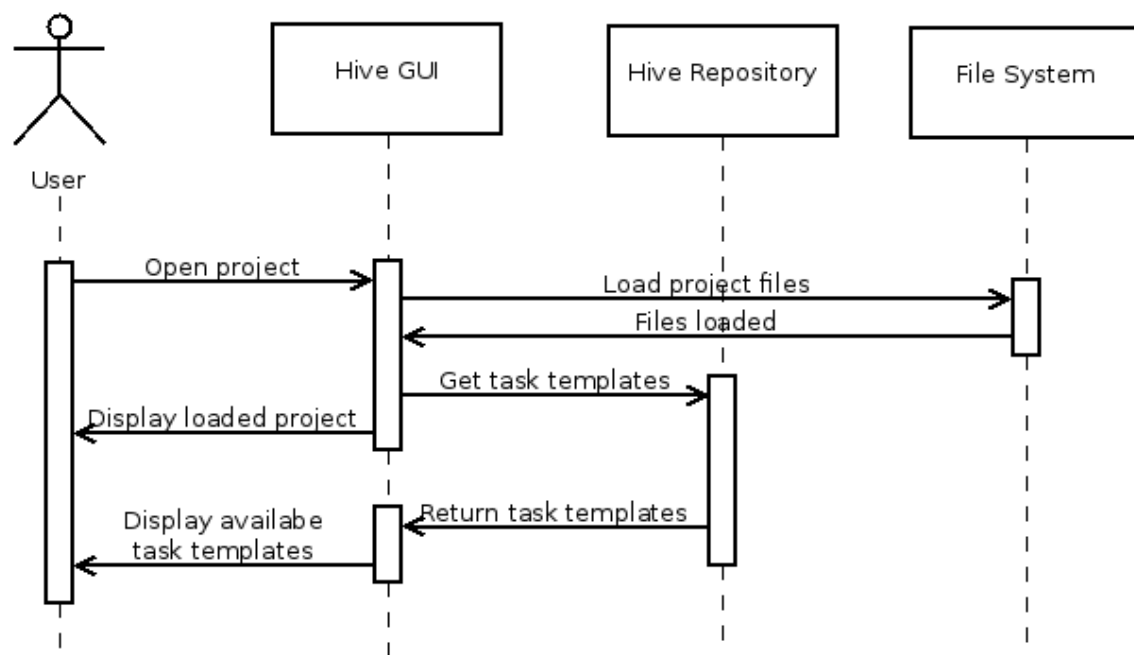


Rysunek 7.3: Diagram utworzenia nowego projektu scenariusza

Otworzenie projektu scenariusza

Zdarzenie to inicjuje Użytkownik, wydając polecenie otwarcia wcześniej utworzonego projektu scenariusza obliczeniowego. Sposób działania systemu można tu podzielić na następujące fazy:

1. Aplikacja uzyskuje od Użytkownika ścieżkę do pliku konfiguracyjnego projektu i wysyła do systemu plików polecenie jego otwarcia
2. System plików lokalizuje żądany plik i zwraca jego zawartość
3. Aplikacja odczytuje plik konfiguracyjny projektu, ładuje do pamięci wszelkie niezbędne do działania dane, i prezentuje je Użytkownikowi - nazwę projektu, strukturę katalogową, graficzny układ zadań obliczeniowych przedstawianych jako węzły grafu
4. Aplikacja komunikuje się z komponentem Repozytorium Szablonów Obliczeniowych, by uzyskać listę szablonów dostępnych do wykorzystania w scenariuszu przez Użytkownika
5. Repozytorium Szablonów Obliczeniowych zwraca listę szablonów dostępnych do wykorzystania
6. Aplikacja ładuje widok listy szablonów obliczeniowych dostępnych do wykorzystania przez Użytkownika

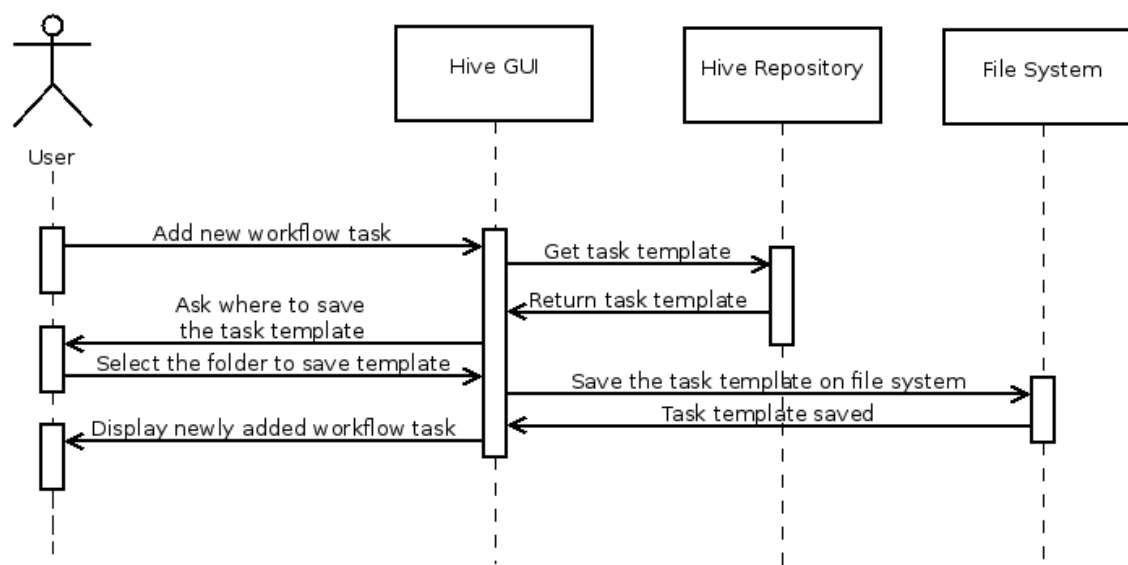


Rysunek 7.4: Diagram otwarcia projektu scenariusza

Dodanie nowego zadania obliczeniowego

Zdarzenie to inicjuje Użytkownik, wybierając jeden z dostępnych szablonów obliczeniowych i przeciągając jego graficzną reprezentację na przestrzeń roboczą projektu scenariusza obliczeniowego. Sposób działania systemu można tu podzielić na następujące fazy:

1. Aplikacja komunikuje się z komponentem Repozytorium Szablonów Obliczeniowych, by pobrać niezbędne dane określające szablon wybrany przez Użytkownika
2. Repozytorium Szablonów Obliczeniowych zwraca Narzędziu Wspomagającemu żądane przez niego dane.
3. Ponieważ główną częścią szablonu obliczeniowego jest modyfikowalny kernel obliczeniowy, Narzędzie Wspomagające pyta Użytkownika, gdzie w systemie plików go zapisać.
4. Użytkownik podaje ścieżkę zapisu nowej instancji szablonu obliczeniowego
5. Aplikacja wydaje do systemu plików polecenie zapisu szablonu obliczeniowego
6. System plików zapisuje szablon obliczeniowy w podanym miejscu i zwraca do Aplikacji komunikat o sukcesie wykonania operacji
7. Narzędzie Wspomagające prezentuje graficzną reprezentację nowo dodanego szablonu obliczeniowego na przestrzeni roboczej projektu scenariusza oraz na panelu ze strukturą katalogową projektu



Rysunek 7.5: Diagram dodawania nowego zadania obliczeniowego

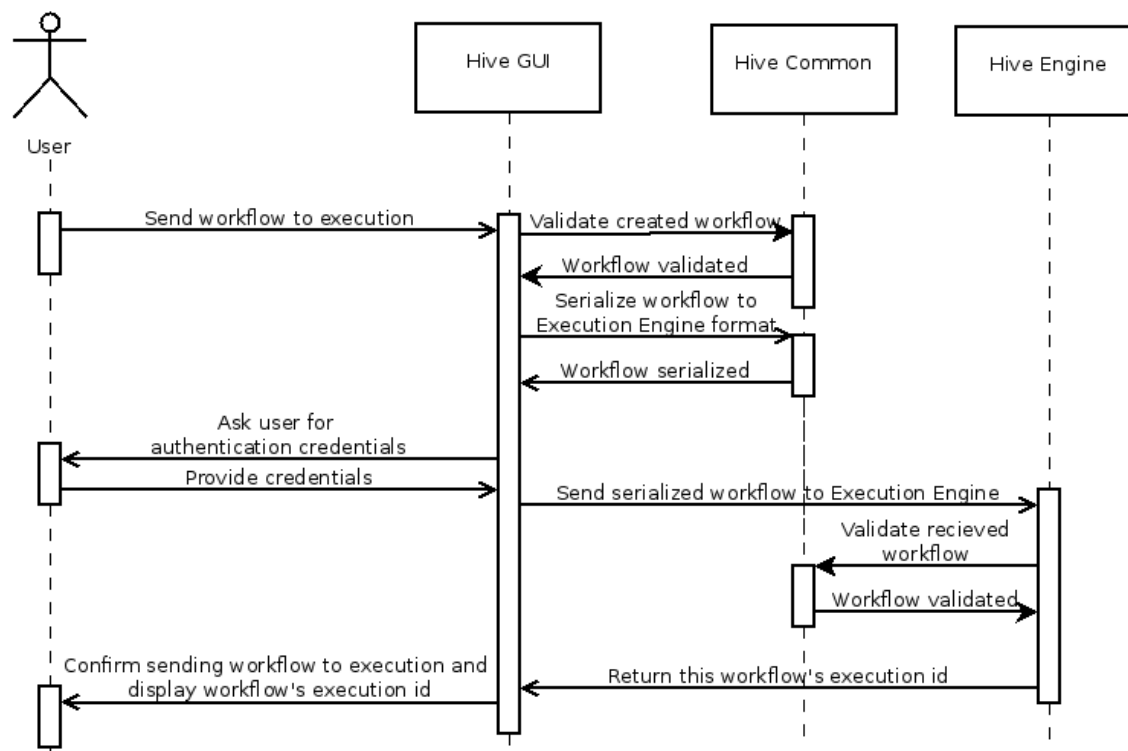
Wysyłania scenariusza obliczeniowego do wykonania

Zdarzenie to inicjuje Użytkownik, wybierając z menu opcję wysłania utworzonego scenariusza obliczeniowego do wykonania w systemie „Kernel Hive”. Sposób działania systemu można tu podzielić na następujące fazy:

1. Aplikacja komunikuje się z komponentem systemowym Common (w którego skład wchodzi Repozytorium Szablonów Obliczeniowych) celem walidacji utworzonego scenariusza - jego rozplanowania, połączeń między zadaniami obliczeniowymi oraz poprawności samych zadań obliczeniowych
2. Komponent Common zwraca wynik walidacji zadanego scenariusza
3. Aplikacja komunikuje się z komponentem Common i wydaje polecenie serializacji scenariusza obliczeniowego do postaci akceptowanej przez Silnik Wykonawczy
4. Komponent Common zwraca scenariusz w zserializowanej formie odpowiadającej formatowi Silnika Wykonawczego
5. Aplikacja pyta Użytkownika o jego dane, wymagane do uwierzytelnienia się w systemie „Kernel Hive”
6. Użytkownik podaje swoje dane uwierzytelniające
7. Aplikacja komunikuje się z Silnikiem Wykonawczym i wysyła scenariusz obliczeniowy do wykonania razem z danymi uwierzytelniającymi Użytkownika
8. Silnik Wykonawczy sprawdza dane uwierzytelniające Użytkownika celem identyfikacji jego tożsamości i uprawnień. Następnie komunikuje się z komponentem systemowym Common celem walidacji otrzymanego scenariusza obliczeniowego
9. Komponent Common zwraca wynik walidacji zadanego scenariusza
10. Silnik Wykonawczy umieszcza scenariusz obliczeniowy w kolejce do wykonania i zwraca do Aplikacji unikalny numer id instancji wykonania scenariusza
11. Aplikacja wyświetla na ekranie potwierdzenie przesłania scenariusza obliczeniowego do wykonania i prezentuje jego numer id

Wyświetlenie listy wykonań scenariuszy obliczeniowych Użytkownika

Zdarzenie to inicjuje Użytkownik, wybierając z menu opcję wyświetlenia listy wykonań swoich scenariuszy obliczeniowych. Sposób działania systemu można tu podzielić na następujące fazy:



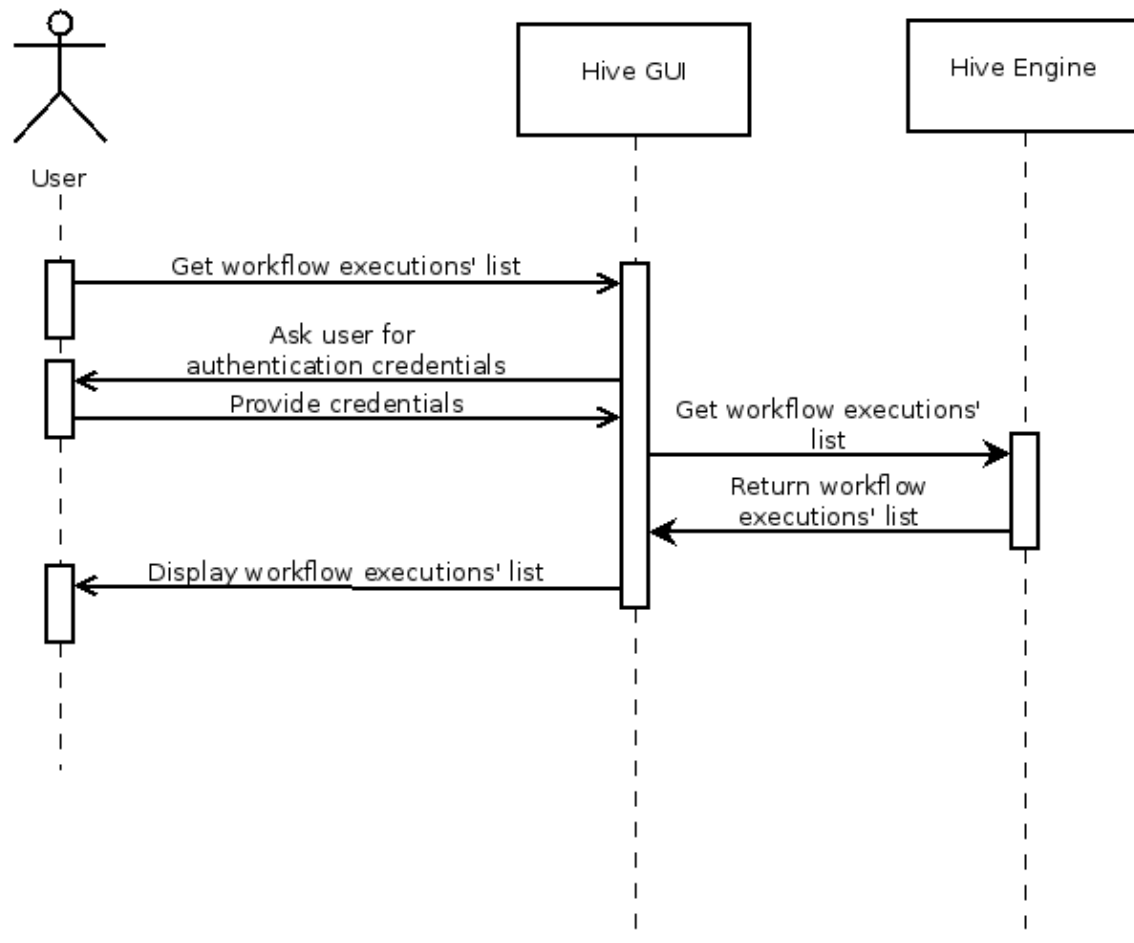
Rysunek 7.6: Diagram wysłania scenariusza do wykonania

1. Aplikacja pyta Użytkownika o jego dane, wymagane do uwierzytelnienia się w systemie „Kernel Hive”
2. Użytkownik podaje swoje dane uwierzytelniające
3. Aplikacja komunikuje się z Silnikiem Wykonawczym i prosi o zwrócenie listy wykonań scenariuszy obliczeniowych danego Użytkownika
4. Silnik Wykonawczy zwraca listę wykonań scenariuszy obliczeniowych Użytkownika
5. Aplikacja wyświetla na ekranie odpowiednio zaprezentowaną żadaną listę wykonań scenariuszy obliczeniowych

Pobranie danych wynikowych z danego wykonania scenariusza obliczeniowego

Zdarzenie to inicjuje Użytkownik, klikając na adresie URL danych wynikowych danego wykonania scenariusza obliczeniowego na widoku listy wykonań scenariuszy obliczeniowych. Sposób działania systemu można tu podzielić na następujące fazy:

1. Aplikacja wydaje polecenie otwarcia domyślnej systemowej przeglądarki internetowej i załadowania zasobów o podanym adresie URL

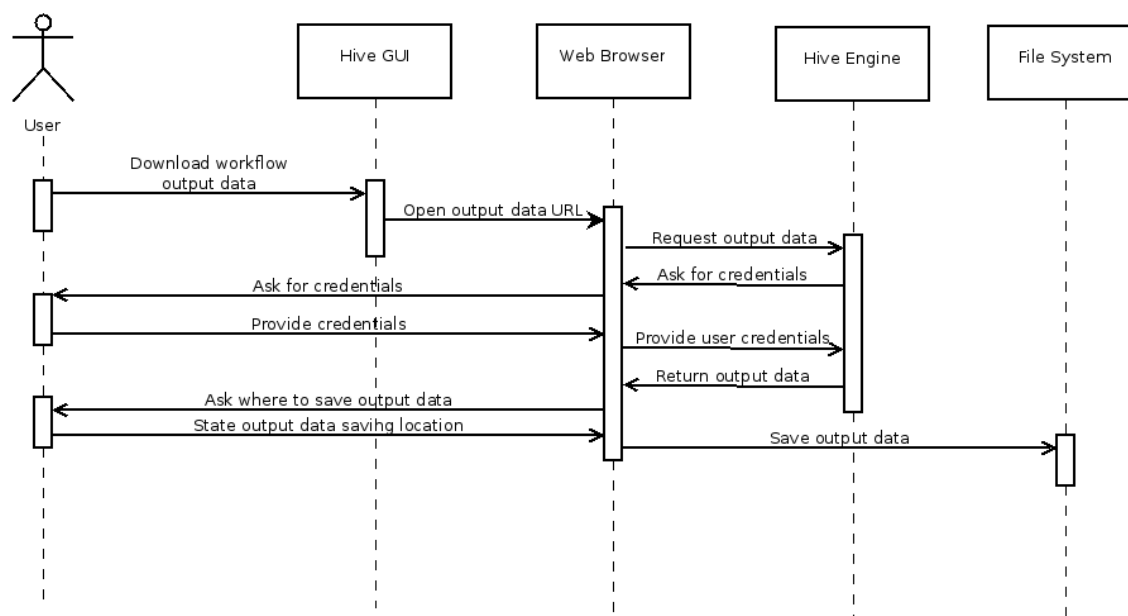


Rysunek 7.7: Diagram wyświetlenia listy wykonanych scenariuszy obliczeniowych Użytkownika

2. Przeglądarka internetowa wykonuje żądanie danego zasobu
3. Silnik Wykonawczy odbiera żądanie i pyta o dane uwierzytelniające konieczne do pobrania zasobu
4. Przeglądarka internetowa prezentuje Użytkownikowi odpowiedni komunikat z prośbą o dane uwierzytelniające
5. Użytkownik wprowadza dane uwierzytelniające
6. Przeglądarka internetowa przekazuje dane uwierzytelniające Użytkownika do Silnika Wykonawczego
7. Silnik Wykonawczy zwraca żądany zasób - dane wynikowe danego wykonania scenariusza obliczeniowego
8. Przeglądarka internetowa pyta Użytkownika, gdzie zapisać pobierane dane

9. Użytkownik wskazuje wybrane przez siebie miejsce w systemie plików

10. Przeglądarka internetowa wysyła do systemu plików polecenie zapisania pobranych danych



Rysunek 7.8: Diagram pobrania danych wynikowych z danego wykonania scenariusza

7.3 Metodologia systemowa

Tutaj opis, w jaki sposób zostało wykonane testowanie/ocena aplikacji

Testowanie systemu jest ważnym etapem w cyklu jego wytwarzania - umożliwia sprawdzenie, czy i w jakim stopniu produkt wynikowy spełnia wymagania postawione przed nim w trakcie fazy określania specyfikacji wymagań systemowych. Z uwagi na to, że Narzędzie Wspomagające jest częścią większej całości, jaką jest system „Kernel Hive”, przeprowadzono dwa rodzaje testów:

- Ocena całości systemu „Kernel Hive”
- Ocenę zaprojektowanego interfejsu użytkownika Narzędzia Wspomagającego

Ocena całości systemu „Kernel Hive”

Celem zbadania działania całości systemu podjęto decyzję o wybraniu przypadku testowego sprawdzającego działanie i stopień integracji poszczególnych komponentów systemu „Kernel Hive”. Zdecydowano się na najbardziej wszechstronny przypadek użycia:

"Użytkownik chce zaprojektować aplikację implementującą równoległy algorytm łamania funkcji skrótu MD5 metodą brutalną, uruchomić ją, a następnie odebrać wynik."

Taki scenariusz testowy daje wiele możliwości oceny systemu w każdym komponencie, jednak w tym wypadku skupiono się na ocenie wydajności systemu - w jaki sposób zmieni się czas wykonania scenariusza w stosunku do zwykłej aplikacji uruchamianej lokalnie na maszynie Użytkownika.

Ocena zaprojektowanego interfejsu użytkownika Narzędzia Wspomagającego

Istnieje wiele metod oceny zaprojektowanego interfejsu:

- Ocena w oparciu o użytkownika (Obserwacje, zapisy dźwiękowe, wywiady, ankiety, zapisy psychofizyczne)
- Ocena w oparciu o teorię (Modelowanie formalne)
- Ocena w oparciu o specjalistę (Ocena heurystyczna, badanie funkcjonalności, przegląd poznawczy)

Podczas oceny interfejsu użytkownika Narzędzia Wspomagającego zdecydowano skoncentrować się na ostatniej grupie i dokonać oceny heurystycznej interfejsu oraz przeglądu poznawczego.

7.4 Wyniki oceny

7.4.1 Wyniki oceny całości systemu „Kernel Hive”

W tej sekcji skupiono się na ocenie całości systemu „Kernel Hive”, pod kątem jego wydajności. Aby miarodajnie określić zysk wydajnościowy, jaki daje nam wykorzystanie systemu przeprowadzono serię testów badających to zagadnienie.

System „Kernel Hive” został poddany testom wydajnościowym, opartych o następujący problem obliczeniowy - łamanie haseł zapisanych za pomocą funkcji MD5 metodą „brute force”. Przyjęto przy tym następujące założenia:

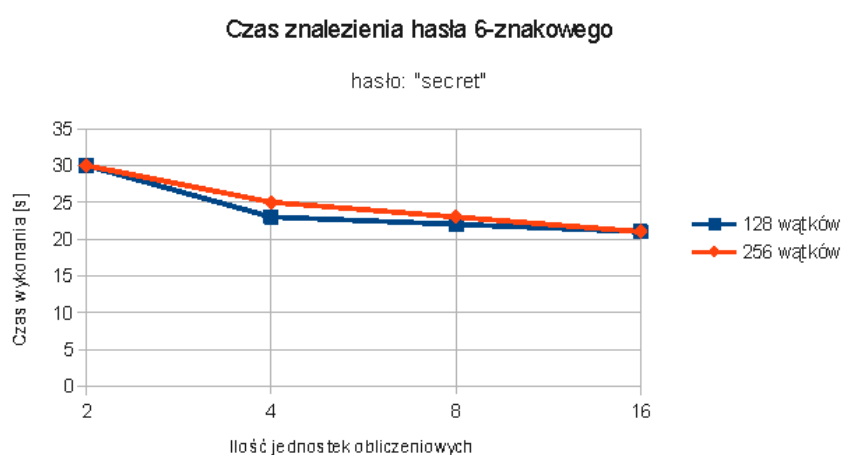
- Hasła składają się tylko z małych liter

Przygotowano odpowiednie paczki danych dla każdego z zestawów testowych oraz hasła, które należy złamać. Przygotowano w ten sposób 3 zestawy testowe:

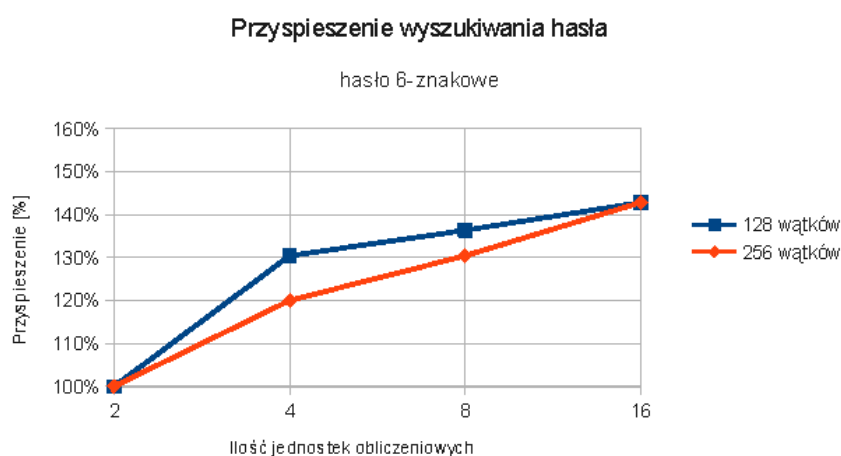
- Hasło 6-znakowe: „secret”
- Hasło 7-znakowe: „khtest”
- Hasło 8-znakowe: „password”

Poniżej przedstawiono wyniki wykonanych testów jako wykresy zależności czasu wykonania programu od ilości jednostek obliczeniowych biorących udział w obliczeniach oraz wykresy zależności przyspieszenia wykonania programu (w procentach) od ilości jednostek obliczeniowych.

1. Hasło 6-znakowe: „secret”

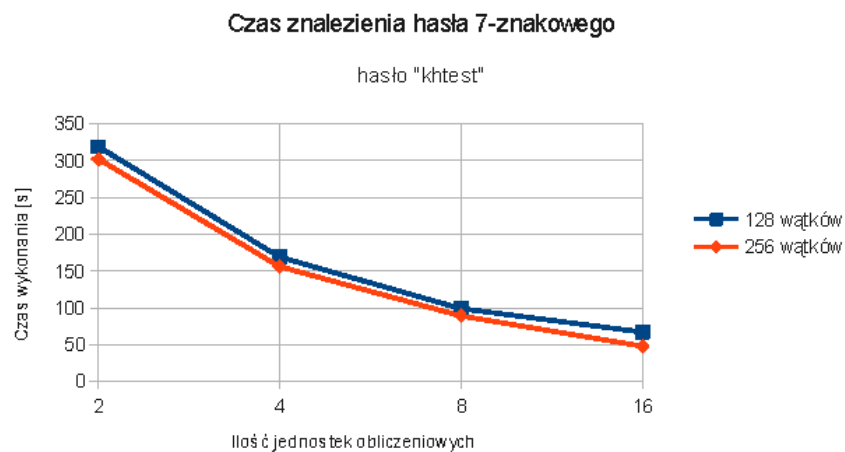


Rysunek 7.9: Wykres czasowy dla hasła 6-znakowego

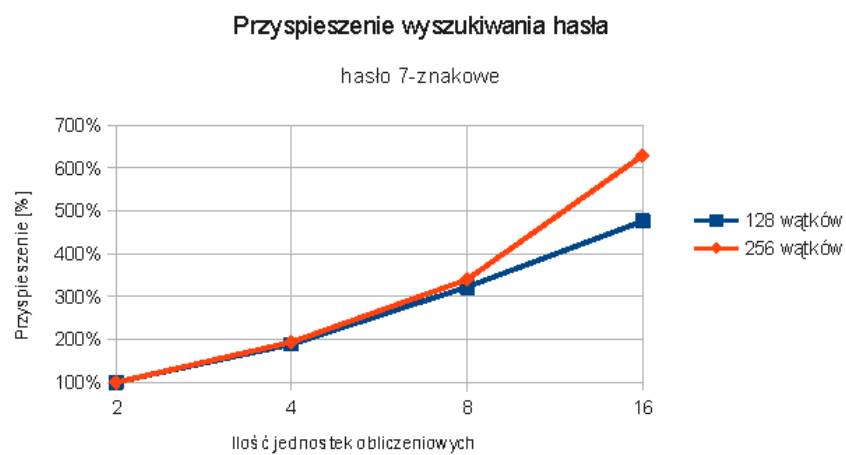


Rysunek 7.10: Wykres przyspieszenia dla hasła 6-znakowego

2. Hasło 7-znakowe: „khtest”

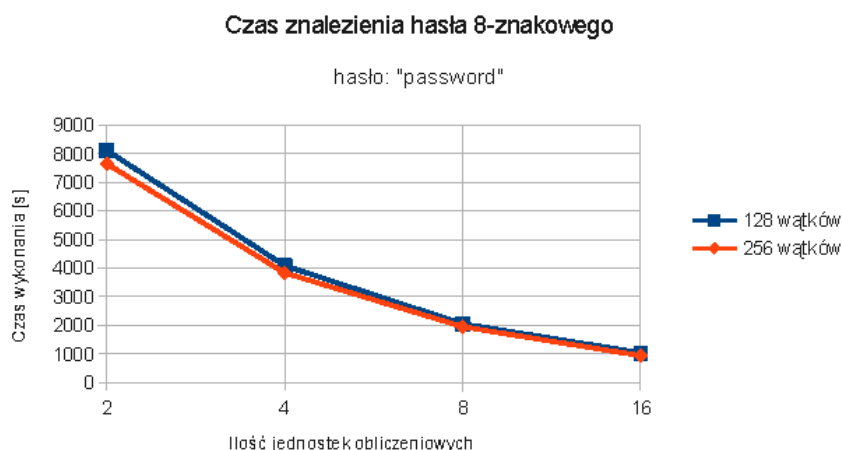


Rysunek 7.11: Wykres czasowy dla hasła 7-znakowego



Rysunek 7.12: Wykres przyspieszenia dla hasła 7-znakowego

3. Hasło 8-znakowe: „password”



Rysunek 7.13: Wykres czasowy dla hasła 8-znakowego



Rysunek 7.14: Wykres przyspieszenia dla hasła 8-znakowego

Jak widać na załączonych wykresach, system istotnie spełnia swoją główną funkcję: przyspiesza działanie aplikacji rozproszonych. Najwyraźniej widać to na podstawie wyników testu z hasłem 8-znakowym - dla danego zakresu testowego, można zauważyć stabilny wzrost wydajności proporcjonalny do liczby wykorzystanych jednostek obliczeniowych.

7.4.2 Wyniki oceny interfejsu użytkownika Narzędzia Wspomagającego

W tej sekcji skupiono się na samej ocenie interfejsu użytkownika Narzędzia Wspomagającego. Do oceny wykorzystano dwie metody - analizę heurystyczną oraz przegląd poznawczy.

Wyniki oceny według analizy heurystycznej

Metoda heurystyczna jest metodą całościową, wykonywaną przez specjalistę lub zespół specjalistów (najlepiej trzech do pięciu), wykorzystującą ich doświadczenie w projektowaniu i ocenie interfejsów. Metoda opiera się na ocenie, jak badany interfejs spełnia poniższe zasady:

1. Widoczny stan systemu

Użytkownik musi wiedzieć, co się dzieje w systemie, w jakim trybie jest interfejs.

Interfejs zmienia swój wygląd w zależności od stanu, w którym się znajduje. Pasek statusu widoczny w dole oka głównego programu dostarcza Użytkownikowi ogólnych informacji o stanie i działaniu aplikacji.

2. Używanie języka użytkownika

Należy używać takich pojęć, które pochodzą ze świata użytkownika, a nie ze świata programisty.

Interfejs używa języka zrozumiałego dla Użytkownika. Nazwy guzików, pól i pozycji menu interfejsu są opisane przy użyciu określeń i wyrażeń zrozumiałych dla ludzi nieposiadających wiedzy o wewnętrznych operacjach zachodzących w systemie „Kernel Hive”.

3. Swoboda działań użytkownika

Użytkownik powinien mieć możliwość swobodnego wyboru kolejności działań i sposobu wykonania zadania. Powinien móc się wycofać z każdej czynności.

Interfejs umożliwia Użytkownikowi przerwanie każdej podjętej czynności. W przypadku wykonywania akcji wieloetapowych, Użytkownik może wycofać się z każdego kroku i zmienić swój wybór w poprzednim.

4. Spójność i używanie standardów

Spójność oznacza, że takie same rzeczy są realizowane w ten sam sposób w różnych miejscach interfejsu, a używanie standardów - że interfejs jest obsługiwany w sposób znany użytkownikowi z innych programów.

Wszystkie standardowe operacje są wykonywane w sposób podobny jak w innych aplikacjach desktopowych. Pozycje w menu zajmują standardowe miejsca, a układ komponentów interfejsu (paski narzędzi, paski statusu, panele boczne) jest podobny do innych programów tego typu.

5. Zapobieganie błędom

zapobieganie błędom polega na uniemożliwieniu użytkownikowi wykonania akcji prowadzącej do błędu.

Program nieustannie monitoruje stan w którym się znajduje, dzięki czemu jest świadomy, jakie akcje mogą być aktualnie wykonane. Akcje inne niż w danym momencie poprawne nie mogą zostać wywołane - interfejs blokuje działanie guzików, pól, pozycji w menu, by zapobiec wprowadzeniu programu przez Użytkownika w niedozwolony stan.

6. Minimalizowanie obciążenia pamięci

Nie należy zmuszać użytkownika do pamiętania informacji o np. składni poleceń, funkcji przycisków, wprowadzonych danych itp.

Guziki i pola w programie opisane są w sposób jednoznacznie określający ich funkcje. Narzędzie Wspomagające zapamiętuje dane wprowadzone przez użytkownika, by przy kolejnym wywołaniu danej akcji możliwe było wybranie wcześniej użytego zestawu danych.

7. Elastyczność i wydajność

Doświadczony użytkownik powinien mieć możliwość jak najszybszego wykonania zadania.

Wszystkie pozycje w menu programu są dostępne za pomocą skrótów klawiszowych. Nawigacja pomiędzy poszczególnymi kartami programu możliwa jest za pomocą klawiatury.

8. Prostota dialogu

Dialog nie powinien zawierać w sobie informacji zbędnej lub rzadko potrzebnej.

Okna dialogowe zawierają w sobie tylko najpotrzebniejsze informacje. Ich treść ich krótka i zrozumiała, daje więc Użytkownikowi konkretną informację o stanie systemu.

9. Dobre diagnozowanie błędów

Dobry komunikat o błędach powinien w sposób dokładny i zrozumiały dla użytkownika poinformować go o problemie i podpowiedzieć możliwe rozwiązanie.

Komunikaty o błędach informują użytkownika w słowach zrozumiałych dla niego.

10. Pomoc i dokumentacja

Należy zapewnić użytkownikowi dodatkowe informacje o działaniu interfejsu.

Dostępny jest podręcznik użytkownika, który pomaga zaznajomić się z możliwościami Narzędzia Wspomagającego. Guziki, pola oraz wpisy w menu o nieoczywistym przeznaczeniu są opisane za pomocą dymków z pomocą kontekstową.

Wyniki oceny według przeglądu poznawczego

W tej metodzie uwaga obserwującego specjalisty skupiona jest na użytkowniku, który wykonuje pojedyncze, wybrane zadanie używając nieznanego mu interfejsu. Użytkownika bada interfejs, by znaleźć działania użyteczne przy wykonywaniu zadania. Wybiera działanie na podstawie wskazówek dostarczanych przez interfejs (etykiety, opisy). Starając się zrozumieć zachowanie systemu, próbuje określić, czy wykonane zadanie zbliżyło go do celu. Wszystkie interakcje użytkownika z interfejsem badane są przez specjalistę.

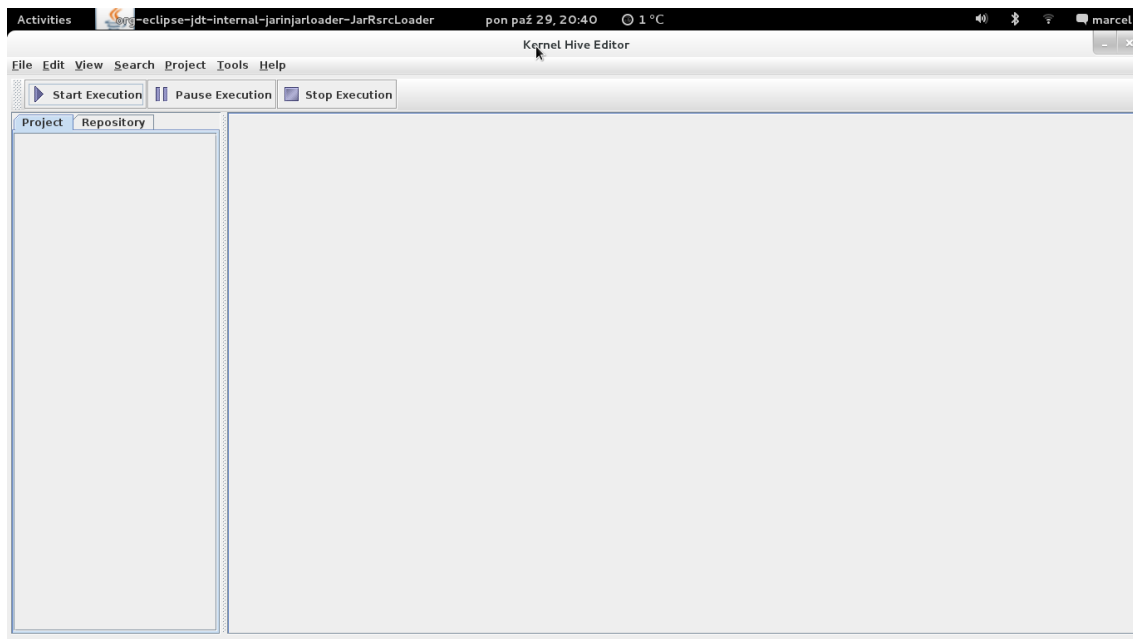
Jako przypadek testowy zostało wybrane zadanie otwarcia utworzonego wcześniej scenariusza obliczeniowego i wysłanie go do wykonania.

Rolę użytkownika wypełniała w tym wypadku jedna z osób biorących udział w rozwoju systemu „Kernel Hive”, niemająca jednak wcześniej styczności z Narzędzie Wspomagającym

Specjalista bada działania użytkownika, sprawdzając w każdym kroku:

1. Czy użytkownik pracuje nad właściwą częścią zadania?
2. Czy użytkownik wie, że właściwe działanie jest dostępne?
3. Czy użytkownik skojarzy cel z właściwym działaniem?
4. Czy użytkownik będzie wiedział, że wykonane działanie przybliżyło go do celu?

Użytkownik pragnie otworzyć wcześniej utworzony scenariusz obliczeniowy i wysłać go do wykonania. Użytkownik wie, gdzie w systemie plików znajduje się katalog z projektem scenariusza.



Rysunek 7.15: Widok głównego okna aplikacji Narzędzie Wspomagające

Użytkownik bada interfejs programu i otwiera **menu *File***.

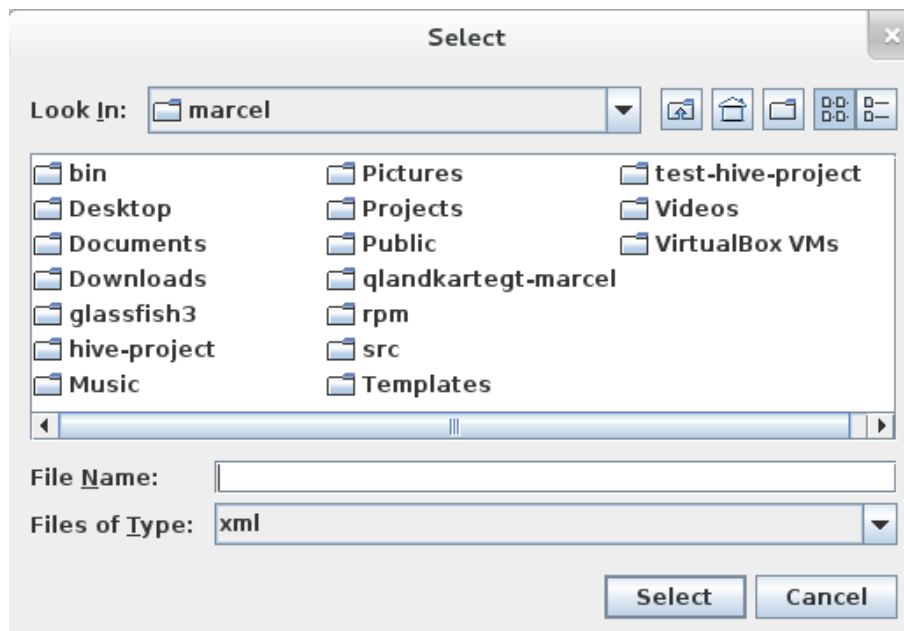
1. Czy użytkownik pracuje nad właściwą częścią zadania?
Tak, zadaniem użytkownika jest utworzenie projektu scenariusza obliczeniowego.
2. Czy użytkownik wie, że właściwe działanie jest dostępne?
Tak, większość aplikacji ma funkcję otwierania plików umieszczoną w menu *File*.
3. Czy użytkownik skojarzy cel z właściwym działaniem?
Tak, skojarzenie z menu *File* wynika z doświadczenia z innymi programami.
4. Czy użytkownik będzie wiedział, że wykonane działanie przybliżyło go do celu?
Tak, w menu *File* występuje pozycja *Open....*

Użytkownik wybiera pozycję *Open....*

1. Czy użytkownik pracuje nad właściwą częścią zadania?
Tak, użytkownik chce otworzyć projekt scenariusza.
2. Czy użytkownik wie, że właściwe działanie jest dostępne?
Tak, nazwa pozycji w menu jednoznacznie to określa.
3. Czy użytkownik skojarzy cel z właściwym działaniem?
Tak, nazwa pozycji w menu jednoznacznie to określa.

4. Czy użytkownik będzie wiedział, że wykonane działanie przybliżyło go do celu?

Tak, pojawia się okno dialogowe otwarcia projektu scenariusza.



Rysunek 7.16: Widok okna dialogowego otwierającego projekt scenariusza

Użytkownik analizuje okno dialogowe otwarcia projektu scenariusza. Przechodzi do odpowiedniego katalogu i otwiera **plik project.xml**.

1. Czy użytkownik pracuje nad właściwą częścią zadania?

Tak, użytkownik chce otworzyć projekt scenariusza.

2. Czy użytkownik wie, że właściwe działanie jest dostępne?

Tak, użytkownik wie to z doświadczenia nabytego przy pracy z innymi programami.

3. Czy użytkownik skojarzy cel z właściwym działaniem?

Tak, okno dialogowe pozwala na otwarcie jedynie plików w formacie XML. W katalogu projektu scenariusza plik `project.xml` jest jedynym, który można otworzyć.

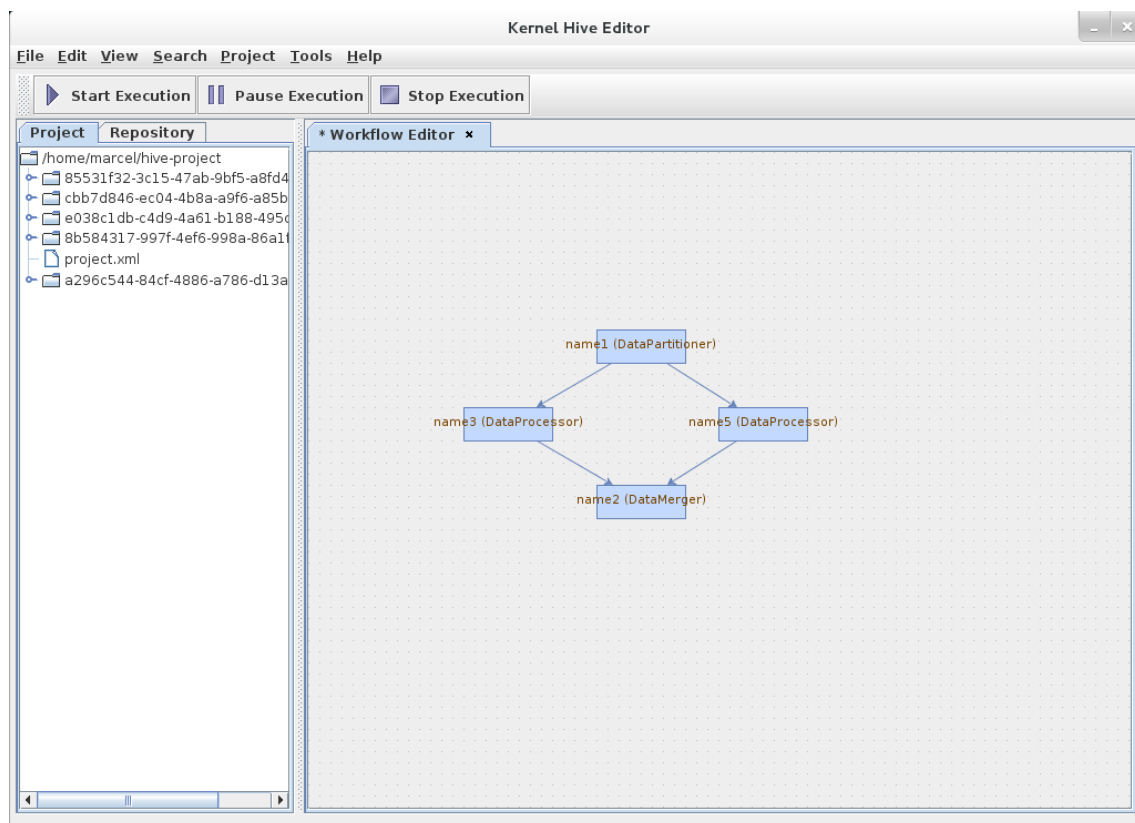
4. Czy użytkownik będzie wiedział, że wykonane działanie przybliżyło go do celu?

Tak, projekt scenariusza zostanie otwarty i załadowany do programu.

Użytkownik bada interfejs programu i naciska guzik ***Start Execution***.

1. Czy użytkownik pracuje nad właściwą częścią zadania?

Tak, użytkownik chce wysłać scenariusz do wykonania.



Rysunek 7.17: Widok głównego okna programu po otwarciu projektu scenariusza

2. Czy użytkownik wie, że właściwe działanie jest dostępne?

Tak, użytkownik widzi aktywny guzik *Start Execution* na głównym pasku narzędzi programu.

3. Czy użytkownik skojarzy cel z właściwym działaniem?

Tak, nazwa guzika jednoznacznie wskazuje na właściwe działanie.

4. Czy użytkownik będzie wiedział, że wykonane działanie przybliżyło go do celu?

Tak, pojawi się okno dialogowe wysłania scenariusza do wykonania.

Użytkownik analizuje okno dialogowe i wybiera naciśnięcie guzika *Next*.

1. Czy użytkownik pracuje nad właściwą częścią zadania?

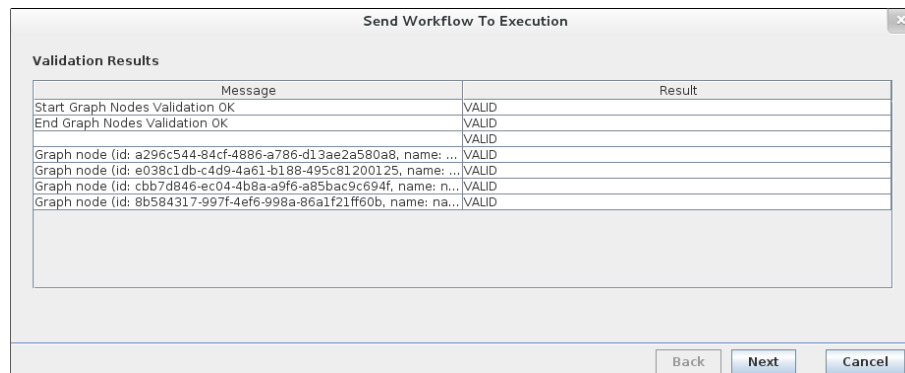
Tak, użytkownik chce wysłać scenariusz do wykonania.

2. Czy użytkownik wie, że właściwe działanie jest dostępne?

Tak, użytkownik widzi aktywny guzik *Next*, a okno dialogowe ma tytuł *Send Workflow To Execution*.

3. Czy użytkownik skojarzy cel z właściwym działaniem?

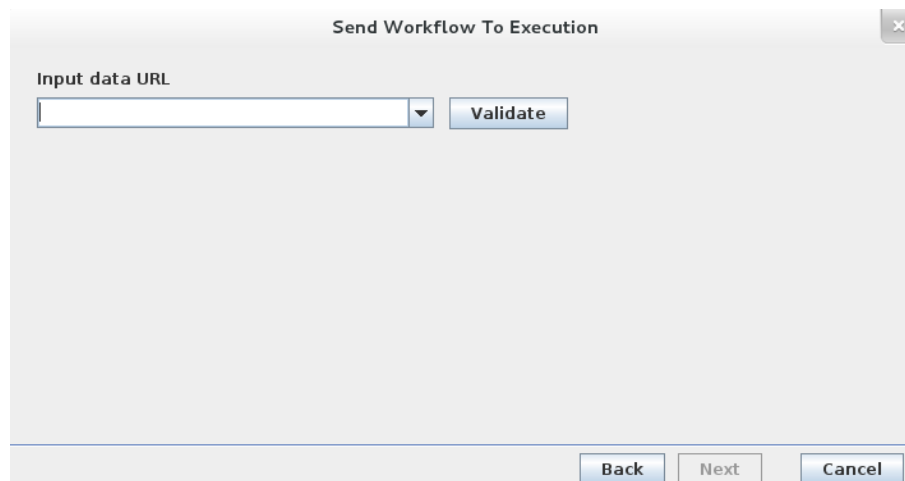
Tak, nazwa guzika *Next* jednoznacznie wskazuje na jego działanie.



Rysunek 7.18: Widok okna dialogowego wysyłającego scenariusz do wykonania - okno wyników walidacji scenariusza

4. Czy użytkownik będzie wiedział, że wykonane działanie przybliżyło go do celu?

Tak, pojawi się druga plansza okna dialogowego wysłania scenariusza do wykonania.



Rysunek 7.19: Widok okna dialogowego wysyłającego scenariusz do wykonania - okno podania adresu danych wejściowych

Użytkownik analizuje okno dialogowe, wpisuje adres URL danych wejściowych dla danego scenariusza i naciska guzik **Validate**. Walidacja adresu URL powiodła się, użytkownik naciska więc guzik **Next**.

1. Czy użytkownik pracuje nad właściwą częścią zadania?

Tak, użytkownik chce wysłać scenariusz do wykonania. Podaje adres URL danych wejściowych i naciska guzik **Validate**. Jeżeli adres URL jest poprawny - guzik **Next** staje się aktywny.

2. Czy użytkownik wie, że właściwe działanie jest dostępne?

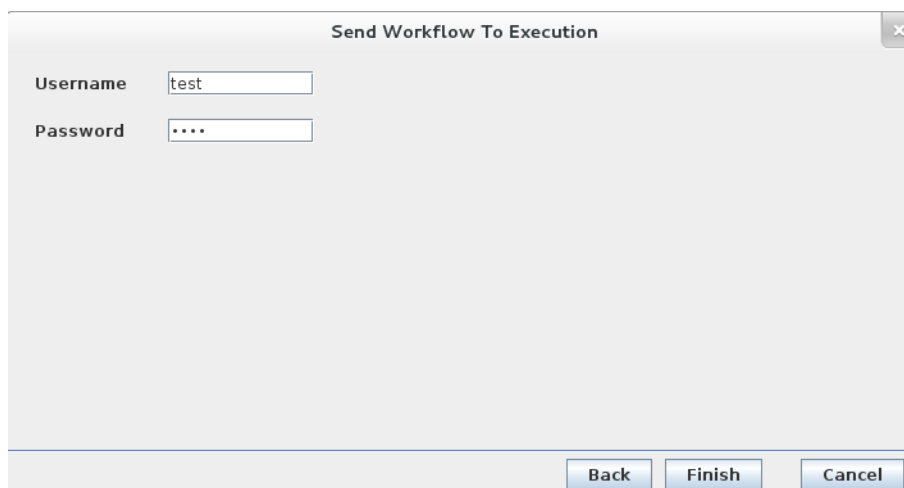
Tak, użytkownik widzi, że guzikiem aktywnym jest guzik **Validate**. Jego naciśnięcie sprawdza poprawność adresu URL i aktywuje guzik **Next**.

3. Czy użytkownik skojarzy cel z właściwym działaniem?

Tak, nazwa guzika *Next* jednoznacznie wskazuje na jego działanie.

4. Czy użytkownik będzie wiedział, że wykonane działanie przybliżyło go do celu?

Tak, pojawi się trzecia plansza okno dialogowego wysłania scenariusza do wykonania.



Rysunek 7.20: Widok okna dialogowego wysyłającego scenariusz do wykonania - okno podania danych uwierzytelniających w systemie „Kernel Hive”

Użytkownik analizuje okno dialogowe, wpisuje swój login i hasło do systemu „Kernel Hive”. Format podanych danych jest poprawny, użytkownik naciska więc guzik ***Finish***.

1. Czy użytkownik pracuje nad właściwą częścią zadania?

Tak, użytkownik chce wysłać scenariusz do wykonania. Podaje swój login i hasło do systemu „Kernel Hive”. Jeżeli dane spełniają warunki poprawności - guzik *Finish* staje się aktywny.

2. Czy użytkownik wie, że właściwe działanie jest dostępne?

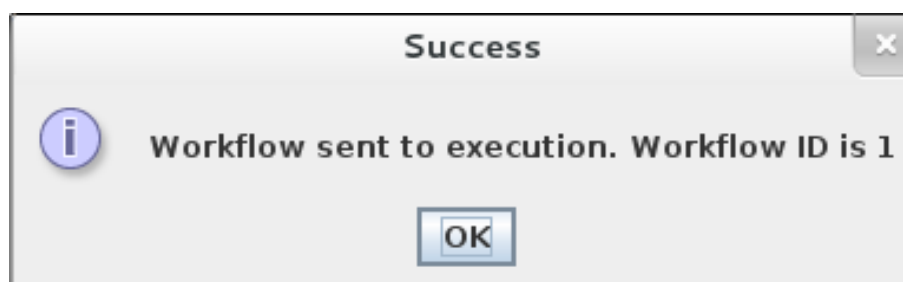
Tak, użytkownik widzi, że wpisanie loginu i hasła aktywuje guzik *Finish*.

3. Czy użytkownik skojarzy cel z właściwym działaniem?

Tak, nazwa guzika *Finish* jednoznacznie wskazuje na jego działanie.

4. Czy użytkownik będzie wiedział, że wykonane działanie przybliżyło go do celu?

Tak, pojawi się okno dialogowe potwierdzające odebranie scenariusza przez Silnik Wykonawczy.



Rysunek 7.21: Widok okna dialogowego potwierdzającego odebranie scenariusza przez Silnik Wykonawczy

7.5 Możliwości rozbudowy

Narzędzie Wspomagające, pomimo, że jest kompletnym rozwiązaniem wspomagającym użytkownika w projektowaniu i uruchamianiu rozproszonych aplikacji równoległych, wciąż umożliwia prace nad ulepszeniami wielu obszarów działania programu. Rozbudowa aplikacji może przyjąć różne postacie, a przykładami mogą tu być następujące kierunki rozwoju:

Graficzna wizualizacja wykonywania scenariusza

Narzędzie może być rozszerzone o moduł odpowiedzialny za graficzną wizualizację aktualnie wykonywanego scenariusza obliczeniowego użytkownika. Ponieważ działanie odbywałoby się w czasie rzeczywistym, użytkownik miałby możliwość podglądu, jaki jest stan zadań obliczeniowych scenariusza - które zostały już wykonane i z jakim skutkiem, a które jeszcze czekają na przetwarzanie.

Integracja Narzędzia jako plugin do środowiska deweloperskiego Eclipse

Eclipse jest jednym z najczęściej używanych środowisk do wytwarzania oprogramowania. Dzięki modułowej budowie, rozszerzalnej dzięki zaawansowanemu mechanizmowi wtyczek możliwe jest dodawanie własnych elementów, wprowadzających nowe funkcje. Dzięki temu możliwym jest zintegrowanie Narzędzia Wspomagającego z Eclipse, uzyskując wtedy kompleksowe środowisko do wytwarzania aplikacji rozproszonych pod postacią scenariuszy. W ten sposób użytkownicy mogliby wykorzystać bogate możliwości Eclipse z zakresu edycji plików z kernelami obliczeniowymi.

Implementacja nowych typów szablonów obliczeniowych

Istnieje wiele zdefiniowanych typów przetwarzania równoległego - "Master-Slave", DAG (ang. *Directional Acyclic Graph*), "Fork-Join", DaC (ang. *Divide and Conquer*), "Map-Reduce" i inne. By zwiększyć uniwersalność systemu i zaoferować użytkownikom jak najwięcej paradygmatów przetwarzania równoległego do wykorzystania, należy rozszerzyć listę dostępnych obecnie szablonów

obliczeniowych o nowe, jeszcze niezaimplementowane typy.

Moduł monitorujący wykorzystanie systemu

Statystyki dotyczące aktualnego działania systemu są bardzo przydatne - mogą wskazywać na duże obciążenie procesora konkretnym zadaniem, wykonywanie zbyt wielu operacji dyskowych itp. Dzięki modułowi monitorowania aktualnego stanu systemu, Użytkownik miałby dostęp do najświeższych danych na temat wykorzystania CPU, GPU, pamięci RAM lub przestrzeni dyskowej.

Rozdział 8

Uwagi końcowe

8.1 Warte podkreślenia osiągnięcia natury informacyjnej

Zaimplementowany w ramach pracy magisterskiej system daje duże możliwości łatwego i intuicyjnego wykorzystania CPU oraz GPU w równoległych aplikacjach rozproszonych. Dzięki temu możliwym jest wykorzystanie całkowitej mocy obliczeniowej wielu klastrów połączonych w sieć gridową do przeprowadzania długotrwałych obliczeń, znacząco skracając czas ich wykonania. Samo Narzędzie Wspomagające oferuje użytkownikowi możliwość prostego graficznego projektowania aplikacji rozproszonych pod postacią scenariuszy. Dzięki temu uwalnia go od obowiązku posiadania wiedzy na temat niskopoziomowych szczegółów komunikacyjnych i pozwala na skupienie się na implementacji logiki biznesowej programu.

8.2 Perspektywy wykorzystania systemu oraz możliwości wdrożeńiowe

Rozważając perspektywy wykorzystania i możliwości wdrożeniowe Narzędzia Wspomagającego, należy rozpatrzyć je jako integralną część większego systemu. Głównym celem przyświecającym powstaniu systemu „Kernel Hive” była demonstracja możliwości, jakie otwiera przed programistami i użytkownikami standard OpenCL, pozwalający na wykonywanie operacji na CPU i GPU w jednolity sposób oraz zaprezentowanie sposobu wykorzystania tego standardu jako podwalin systemu oferującego narzędzia do tworzenia rozproszonych aplikacji równoległych w wysoce abstrakcyjny sposób.

Mając powyższe założenia na uwadze, można skupić się na następujących funkcjach systemu:

1. Funkcja edukacyjna

System, z racji swojej łatwości i przejrzystości w użyciu może służyć jako platforma edukacyjna, pozwalająca na wprowadzenie studentów w zagadnienia programowania równoległych aplikacji z wykorzystaniem standardu OpenCL. Dzięki zachowaniu wysokiego stopnia abstrakcji podczas projektowania aplikacji jako scenariuszy obliczeniowych, uczące się osoby mogą skupić się na samym tworzeniu kerneli OpenCL, pomijając na razie kwestie komunikacyjne.

2. Funkcja badawcza

Jednym z celów systemu „Kernel Hive” była demonstracja możliwości standardu OpenCL do wykorzystania mocy obliczeniowej CPU i GPU podczas wykonywania aplikacji rozproszonych. Udało się osiągnąć to zamierzenie, co sprawia, że bazujący na tym rozwój systemu może pójść w innych, nie mniej interesujących kierunkach - równoważenie obciążenia, odporność na awarie, inteligentne wykorzystanie zasobów itp.

3. Funkcja biznesowa

System „Kernel Hive” został pomyślany jako platforma dla użytkowników nieposiadających wykształcenia informatycznego z zakresu projektowania aplikacji rozproszonych. Dzięki zapewnieniu komunikacji pomiędzy poszczególnymi komponentami systemu oraz dostarczeniu użytkownikowi sposobu na łatwe i intuicyjne tworzenie aplikacji w formie scenariuszy, pozwala na skupieniu się na implementacji logiki biznesowej, pomijając niskopoziomowe zagadnienia z zakresu przesyłu komunikatów i danych.

Biorąc pod uwagę powyższe funkcje systemu „Kernel Hive” można stwierdzić, że znalazłby wykorzystanie jako platforma edukacyjna, wzbogacająca laboratorium Katedry Architektury Systemów Komputerowych. Jednoczesny rozwój oprogramowania przyczyniłby się do powstania kompleksowego produktu, który byłby w stanie zainteresować klientów biznesowych.

Rozdział 9

Bibliografia

Rozdział 10

Załączniki

Lista załączników:

- Diagram klas
- Diagram sekwencji głównych akcji w systemie
- Podręcznik użytkownika