AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

**WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI,
INFORMATYKI I INŻYNIERII BIOMEDYCZNEJ**

KATEDRA INFORMATYKI STOSOWANEJ

Praca dyplomowa magisterska

Analiza możliwości adaptacji środowisk uruchomieniowych dla logiki biznesowej na platformach mobilnych.

Analysis of possible adaptation or business logic runtimes for mobile platforms.

Autor: *Tomasz Landowski*
Kierunek studiï£¡w: *Informatyka*
Opiekun pracy: *dr hab. inż. Grzegorz J. Nalepa*

Kraków, 2014

Oświadczam, świadomy(-a) odpowiedzialności karnej za poświadczenie nieprawdy, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

Serdecznie dziękuję Panu dr hab. inż. Grzegorz J. Nalepa, za poświęcony czas i cenne wskazówki, które przyczyniły się do powstania niniejszej pracy magisterskiej.

Spis treści

1. Wstęp

Wzrost popularności aplikacji mobilnych głównie ze względu na powszechny dostępu do urządzeń przenośnych takich jak smartphone czy tablet, wpłynął na wzmocnienie ich pozycji na rynku IT. W ostatnim czasie powstał szereg firm zajmujących się tworzeniem tylko tego rodzaju systemów informatycznych. Przyglądając się współczesnym trendom w IT nie sposób więc nie zwrócić szczególnej uwagi właśnie na aplikacje mobilne. Mimo wcześniej wspomnianego wzrostu popularności systemy mobilne ciągle wymagają od programistów korzystania z niskopoziomowych bibliotek i narzędzi. Ze względu na rodzaj problemów rozwiązywanych przez tego typu aplikacje może to być częściowo zrozumiałe. Aplikacje mobilne jednak to nie tylko proste programy uruchamiane w ograniczonym sprzętowo środowisku. Twórcy tego rodzaju aplikacji muszą się zmagać z problemami synchronizacji danych, rozproszonej architektury itd. Z drugiej zaś strony wyżej wymienione problemy są już od dawna znane, co więcej posiadają wypracowane schematy ich rozwiązywania. W przypadku aplikacji internetowych powstało mnóstwo frameworków i narzędzi wspierających tworzenie tego rodzaju systemów. Szablonowe rozwiązania pozwoliły natomiast na przeniesienie rozwiązywania problemów na znacznie wyższy poziom niż pisanie kodu aplikacji. Powstały w ten sposób różnego rodzaju systemy uruchomieniowe logiki biznesowej, skalowane i łatwo modyfikowalne. Niniejsza praca magisterska jest próbą wykorzystania istniejących środowisk uruchomieniowych dla logiki biznesowej w środowisku aplikacji mobilnych.

1.1. Cel pracy

Celem niniejszej pracy magisterskiej jest zestawienie technologii modelowania logiki biznesowej z aplikacjami mobilnymi. Powinno ono prowadzić do wyciągnięcia wniosków dotyczących sensowności oraz sposobu wykorzystania tych dwóch kompletnie różnych technologii razem.

1.2. Struktura pracy

W rozdziale 2 została zawarta analiza modelowania procesów biznesowych w kontekście aplikacji mobilnych. Rozdział został podzielony na cztery sekcje, które zawierają kolejno:

– Sekcja 2.1 - Opis technologii modelowania procesów biznesowych.

– Sekcja 2.2 - Opis specyfiki aplikacji na platformy mobilne.

– Sekcja 2.3 - Opis BPEL - jednej z implementacji BPM w środowisku usług sieciowych.

– Sekcja 2.4 - Zestawienie BPEL z aplikacjami mobilnymi.

Rozdział 3 jest opisem projektu warstwy pośredniej, której zastosowanie zostało zaproponowane w rozdziale poprzednim. Rozdział rozpoczyna się od przedstawienia bardzo poglądowych diagramów projektu, by następnie poprzez kolejne fazy projektowania rozwiązana przejść do bardziej szczegółowych diagramów klas i sekwencji.

Zwieńczeniem pracy magisterskiej jest rozdział 4, który zawiera kompleksowy przykład zastosowania przedstawionego wcześniej rozwiązania. Rozdział zawiera poglądowy diagram BPMN implementowanego procesu biznesowego i opis czterech podsystemów niezbędnych do implementacji tego procesu.

2. Zestawienie modelowania procesów biznesowych ze specyfiką aplikacji mobilnych

Tematem niniejszej pracy magisterskiej jest analiza możliwości adaptacji środowisk uruchomieniowych dla logiki biznesowej na platformach mobilnych. Temat jest szeroki, zarówno środowiska uruchomieniowe dla logiki biznesowej jak i platformy mobilne mogą być rozumiane w różny sposób, dlatego na początku niezbędne jest uściślenie kilku kwestii. Środowiska uruchomieniowe dla logiki biznesowej w niniejszej pracy magisterskiej rozumiane będą przez technologię Modelowania Procesów Biznesowych. W przypadku platform mobilnych największa uwaga zostanie poświęcona najbardziej popularnej, opartej na systemie operacyjnym *Unix* platformie mobilnej - *Android*. Aby przejść do zestawienia tych dwóch technologii niezbędne jest ich poznanie i zrozumienie, dlatego w kolejnych sekcjach niniejszego rozdziału zostaną opisane wyżej wymienione technologie.

2.1. Modelowania Procesów Biznesowych

Modelowanie Procesów Biznesowych jest pojęciem bardzo ogólnym, nie zawiera w sobie żadnych szczegółów technicznych, określa jedynie pewne specyficzne podejście do rozwiązywania problemów informatycznych. W podejściu tym podczas analizy, główny nacisk kładziony jest na wyłonienie procesów występujących w analizowanym problemie. Proces jest tutaj rozumiany jako zbiór następujących w określonej kolejności operacji, prowadzących do osiągnięcia konkretnego celu. Najczęściej modelowanie procesów biznesowych jest rozważane w kontekście działania konkretnego przedsiębiorstwa, w którym wyłonienie procesów oraz odpowiednie nimi zarządzenie ma kluczowe znaczenie dla osiągnięcia sukcesu.

2.1.1. BPM

W literaturze pojęcie modelowania oraz zarządzanie procesami biznesowymi w kontekście przedsiębiorstw występuje pod skrótem BPM, którego angielskie rozwinięcie to *Buisness Process Management* (pl. Zarządzanie Procesami Biznesowymi). Idea BPM jest bardzo popularna i szeroko rozwijana w środowisku IT, powstała nawet organizacja pod nazwą *European Association of Buisness Process Managment* zajmująca się rozwojem i promocją BPM. Na stronie internetowej organizacji znaleźć możemy oficjalną definicje BPM, która mówi, że w skład Zarządzania Procesami Biznesowymi wychodzi:

– projektowanie,

– wykonywanie,

– dokumentacja,

– pomiar,

– monitorowanie,

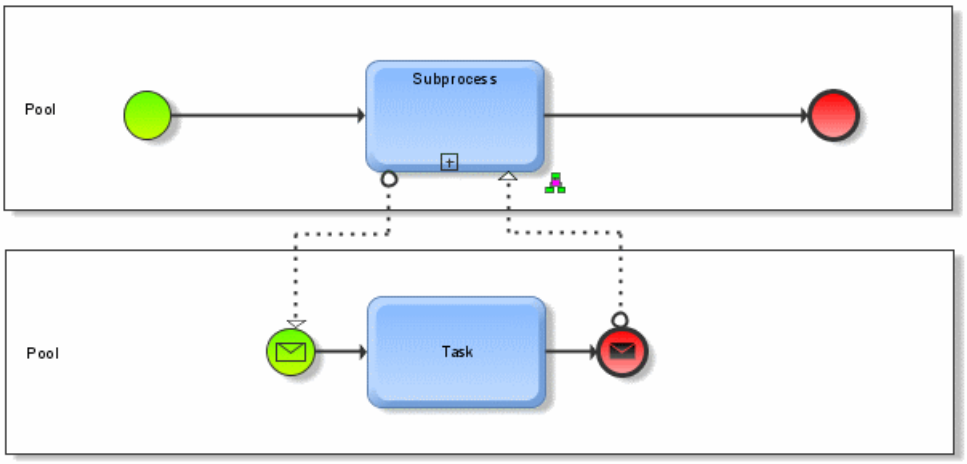
– kontrola

zautomatyzowanych oraz niezautomatyzowanych procesów biznesowych [1].

W środowisku IT zdarza się, że BPM traktowany jest jako osobna klasa systemów informatycznych, obok ERP, MES, CRM itd. Z punktu widzenia przedstawionej wyżej definicji trudno zgodzić się z takim podejściem, można jednak zauważyć że BPM de facto może zostać wykorzystany do realizacji każdego z wymienionych rodzajów oprogramowania lub posłużyć jako narzędzie integrujące wyżej wymienione klasy systemów, w celu stworzenia globalnego systemu zarządzania przedsiębiorstwem [2].

2.1.2. Projektowanie procesów biznesowych

Zdecydowanie jednym z najistotniejszych etapów w tworzeniu systemu opartego o procesy biznesowe jest etap projektowania. Jednym z najbardziej popularnych narzędzi do tego celu jest BPMN (*Buisness Process Model and Notation*), które doczekało się już dwóch wersji. W kontekście niniejszej pracy magisterskiej notacja BPMN nie jest szczególnie ważna jednak została wspomniana ze względu na jej wykorzystanie w przykładach. BPMN z jednej strony jest narzędziem do graficznego modelowania procesów biznesowych, jest to pewien standard, który z założenia ma być intuicyjny dla osób z poza środowiska IT. Z drugiej zaś strony istnieją narzędzia pozwalające za pomocą BPMN modelować procesy które są bezpośrednio uruchamiane na konkretnych platformach [7].



Rysunek 2.1: Przykładowy proces BPMN.

2.1.3. Wykonywanie procesów biznesowych

Jak zostało wspomniane na samym początku Modelowanie Procesów Biznesowych jest pojęciem ogólnym nie wskazującym konkretnej technologii, która ma posłużyć do wykonania procesów biznesowych. Z definicji można jednak wnioskować, że środowiskiem do wykonywania procesów biznesowych w przedsiębiorstwach na pewno nie powinna być prosta aplikacja desktopowa czy mobilna. Biorąc pod uwagę wachlarz zastosowań BPM nie trudno dojść do wniosku, że najbardziej odpowiednim środowiskiem dla procesów biznesowych są środowiska aplikacji rozproszonych. Powyższe stwierdzenie potwierdza przegląd istniejących systemów uruchomieniowych dla procesów biznesowych. W ogromnej większości są to aplikacje internetowe.

2.1.4. Podsumowanie

Z punktu widzenia niniejszej pracy magisterskiej, głównym wnioskiem płynącym z powyższego opisu BPM jest skala problemów rozwiązywanych przez to podejście oraz ich usytuowanie w środowisku aplikacji rozproszonych.

2.2. Analiza aplikacji na platformy mobilne

Wzrost popularności urządzeń przenośnych takich jak smatrphon czy tablet, wpłynął na powstanie specyficznej klasy systemów informatycznych, zwanych aplikacjami mobilnymi. Aplikacje mobilne cechuje ukierunkowanie na rozwiązywanie wąskiego rodzaju problemów przy pomocy urządzenia dostępnego dla użytkownika niemal 24h na dobę. Szczególną popularność zyskały aplikacje mobilne ukierunkowane na rynek globalny tzn. dla użytkownika masowego. Coraz większą popularność zyskują również aplikacje biznesowe ukierunkowane np. na wymianę danych pomiędzy pracownikami przedsiębiorstwa w celu rozwiązania konkretnego zadania.

2.2.1. Rodzaje aplikacji mobilnych

Ze względu na rodzaj zastosowania aplikacje mobilne można sklasyfikować w następujący sposób:

– Samodzielne aplikacje – aplikacje wykorzystujące tylko i wyłącznie lokalne zasoby urządzenia, korzystające z lokalnej bazy danych bez połączenia z systemami zewnętrznym, często nie wymagające połączenie do internetu. Dobrym przykładem takiej aplikacji może być prosty notatnik.

– Aplikacje klienckie – są to aplikacje bazujące na komunikacji z systemami zewnętrznymi przy pomocy jakiegokolwiek interfejsu komunikacyjnego, najczęściej połączenia HTTP. Zazwyczaj aplikacje te wykorzystują również lokalną bazę danych umożliwiając użytkownikom pracę z aplikacją w przypadku braku połączenia z systemem zewnętrznym.

– Internetowe – najczęściej strony www, aplikacje te nie wykorzystują lokalnych zasobów urządzeń mobilnych, ich rolą jest udostępnienie prostego interfejsu użytkownika systemu dla użytkowników aplikacji mobilnych.

– Gry – szczególny przypadek samodzielnych aplikacji ukierunkowanych głównie na wykorzystanie lokalnych zasobów urządzenie w celach rozrywkowych, szczególnie eksploatowana w tego typu aplikacjach jest karta graficzna urządzenia.

2.2.2. Tryby online/offline

Najbardziej docenianym rodzajem aplikacji mobilnych z punktu widzenia biznesu są aplikacje klienckie. W przypadku tych aplikacji bardzo istotny jest problem połączenia z systemami zewnętrznymi. Wyróżniamy tutaj dwa tryby pracy aplikacji:

– Tryb online – urządzenie udostępnia przynajmniej jeden typ komunikacji bezprzewodowej (Wi-fi, Bluetooth, łącza podczerwieni (IrDa), GPRS). W trybie online aplikacje mobilne mają bezpośredni dostęp do zewnętrznych i zdalnych źródeł danych, innych urządzeń mobilnych lub stacjonarnych systemów komputerowych.

– Tryb offline – urządzenie ma bezpośredni dostęp tylko do lokalnie przechowywanych informacji. Dane te mogą być jednak synchronizowane z innymi urządzeniami w czasie krótkich sesji komunikacyjnych. Wymiana danych podczas synchronizacji może następować w obu kierunkach.

2.2.3. Cechy specyficzne dla aplikacji mobilnych

Opisując aplikacje mobilnie nie można zapomnieć o cechach specyficznych, które są niezwykle ważne z punktu widzenia projektanta i programisty aplikacji mobilnych. To właśnie dzięki tym cechom aplikacje mobilne posiadają tę, a nie inną specyfikę:

– Ograniczone zasoby sprzętowe – urządzenie przenośnie pod względem zasobów sprzętowych nigdy nie zastąpią komputerów stacjonarnych, a tym bardziej urządzeń serwerowych. Właśnie ze względu na tę cechę aplikacje mobilne ukierunkowane są na rozwiązywanie małych - jednostkowych problemów. Projektant aplikacji mobilnych powinien szczególnie zwrócić uwagę na ten aspekt w przypadku projektowania interfejsu do komunikacji z systemami zewnętrznymi. Asplikacje mobilne powinny wykorzystywać lekkie interfejsy komunikacyjne np. REST, aby jak najmniej obciążać kartę sieciową urządzenia.

– Uproszczony interfejs użytkownika dostosowany do ekranów dotykowych – cecha ta jest bardzo ważna głównie dla projektantów graficznych, ale ma również znaczący wpływ na rodzaj zadań rozwiązywanych za pomocą aplikacji mobilnych.

– Przerwy w działaniu aplikacji – aplikacje mobilne w przeciwieństwie do innych rodzajów systemów są szczególnie narażone na przerwy w działaniu. Najprostszym przykładem może być roz-

mowa telefoniczna przychodząca w trakcie pracy z aplikacją. Najczęściej w takich przypadkach aplikacje przechodzą w tryb uśpienia i nie ma pewności czy praca zostanie wznowiona. Programiści powinny pamiętać o zapamiętywaniu poszczególnych stanów aplikacji aby użytkownik nie tracił wykonanej pracy.

– Dostęp do geolokalizacji – aplikacje mobilne oprócz ograniczeń posiadają również wiele cech dodatkowych, wyróżniających je wśród innych systemów - jedną z nich jest dostęp do geolokalizacji. Funkcjonalność ta bardzo szeroko wykorzystywana jest przez systemy zwane *Context Aware Middleware* [3].

– Dostęp do aparatu/kamery – jest to kolejna cecha rozszerzająca możliwości aplikacji mobilnych, może być wykorzystana na przykład jako narzędzie do skanowania kodów kreskowych, albo sposób dokumentacji pracy wykonanej przez użytkownika systemów mobilnych.

– Specyficzna architektura dla różnych systemów operacyjnych – przyglądając się najbardziej popularnym platformą mobilnym można odnaleźć bardzo wiele cech współnych, mimo wszystko platformy te korzystają zazwyczaj z różnych technologii do realizacji podobnych celów.

2.2.4. Komunikaty Push

Ze względu na zmienne warunki w dostępie do mediów komunikacyjnych, aplikacji mobilne wykorzystują w głównej mierze komunikacje typu pull (pobieranie informacji na żądanie) do wymiany informacji z systemami zewnętrznymi. Najpopularniejsze platformy mobilne jak *Android*, *iOS* oraz *Windows Phone* udostępniają również alternatywny sposób komunikacji - push.

Komunikacja push odbywa się jednak zawsze za pośrednictwem serwisów udostępnianych przez platformy mobilne do tego celu. Komunikacja ta posiada wiele ograniczeń, a już na pewno najważniejszym z nich jest brak pewności, że komunikat zostanie dostarczony. Komunikacja push powinna być zatem wykorzystywana jedynie do notyfikacji użytkownika o jakimś zdarzeniu, a nie do przesyłania informacji kluczowych dla działania aplikacji.

2.3. BPEL „inteligentne" narzędzie integracyjne

BPEL (*Business Process Execution Language*), a właściwie pełna nazwa WS-BPEL, której rozwinięcie to *Web Services Business Process Execution Language* jest językiem opartym o składnię xml, służącym do wykonywania procesów biznesowych w środowisku usług sieciowych. Język BPEL jest ustandaryzowany przez oranizacje OASIS (*Organization for the Advancement of Structured Information Standards*).

Organizacja OASIS definiuje język BPEL jako: *Umożliwiający użytkownikom opis aktywności procesów biznesowych jako usługi sieciowe oraz definicję w jaki sposób usługi te mogę być połączone między sobą w celu wykonania zadania* [4].

Jak argumentuje OASIS, język BPEL powstał z konieczności stworzenia dodatkowej warstwy integracyjnej, która wykorzysta potencjał dostarczony przez usługi sieciowe oraz procesy biznesowe. Warto tutaj wspomnieć o historii powstania BPEL. Microsoft i IBM dostrzegając potrzebę stworzenia technologii pozwalającej definiować przepływ wywołań usług sieciowych stworzyli osobne, ale bardzo podobne języki nazwane WSFL (*Web Service Flow Language*) oraz Xlang. W miare wzrostu popularności BPM zdecydowali się połączyć siły aby stworzyć wspólny standard, opisanych za pomocą WSDL. Zarówno Microsoft jak i IBM dostrzegli, że architektura SOA (*Service Oriented Architecture*) wymaga wprowadzenia technologi, która pozwoli składać klocki jakimi są usługi sieciowe w jedną zgrabną całość.

Dotychczasowy model komunikacji udostępniany przez usługi sieciowe był niewystarczający, głównie z powodu braku zachowania stanu podczas prostej komunikacji żądanie-odpowiedź. Komunikacja ponadto była głównie jednokierunkowa. Model komunikacji dla biznesu wymagał natomiast sekwencyjnej wymiany wiadomości pomiędzy wieloma węzłami. Operacje wykonywane na węzłach mogły trwać bardzo długo, dlatego zainstaniała konieczność obsługi tego typu sytuacji przy pomocy asynchronicznego wywoływania serwisów [5].

2.3.1. Cechy języka BPEL

- Definiowane procesy biznesowe komunikują się z usługami sieciowymi przy pomocy WSDL 1.1 i same są usługami sieciowymi opisanymi przez WSDL 1.1.
- Do komunikacji pomiędzy usługami wykorzystywany jest protokół SOAP.
- Procesy zdefiniowane są w języku bazującym na xml.
- BPEL dostarcza funkcji do wykonywania prostych manipulacji na danych.
- Posiada wsparcie do identyfikowania instancji procesów.
- Posiada wsparcie dla podstawowego cyklu życia procesu.
- Wspiera transakcyjność przy wykorzystaniu podstawowych technik takich jak akcje kompensacyjne.

2.3.2. Orkiestracja czy choreografia?

BPEL jest językiem orkiestracji usług sieciowych, a nie choreografii. Podstawowa różnica polega na tym, że w choreografii usługi współpracują ze sobą bezpośrednio. W orkiestracji usługa komunikuje się wyłącznie z menedżerem orkiestracji (w tym przypadku: maszyna BPEL), który wie, gdzie i jak przekazać dalej komunikaty. Nie musi tym samym znać innych usług biorących udział w procesie.

Korzyści płynące z tego, że BPEL jest warstwa orkiestracji to [6]:

- podejście integracyjne do komunikacji pomiędzy aplikacjami, które uniezależnia aplikacje od siebie,

– umożliwia sterowanie przepływem, zarządzanie bezpieczeństwem oraz niezawodność komunikacji,

– co najważniejsze sposób zarządzanie i monitorowania procesów jest scentralizowany.

2.3.3. Silniki uruchomieniowe WS-BPEL

WS-BPEL dostarcza jedynie opisu w jaki sposób proces ma przebiegać oraz z jakimi usługami ma się komunikować, potrafi również zdefiniować proste operacje proceduralne jak tworzenie zmiennych, operacje przypisania, warunki czy pętle. Język sam w sobie jednak nie ma zbyt wielkiej wartości bez odpowiedniego środowiska uruchomieniowego. Procesy BPEL można traktować jako skrypty interpretowane, wykonywane przez maszyny procesów biznesowych w środowisku zgodnym ze standardem BPEL.

Najbardziej popularne środowiska uruchomieniowe BPEL to:

– *ActiveVOS*

– *Apache ODE*

– *ExpressBPEL BPM*

– *BizTalk Server*

– *Open ESB*

– *Oracle BPEL Process Manager*

– *OW2 Orchestra*

– *Parasoft BPEL Maestro*

– *Petals BPEL Engine*

– *WebSphere Process Server*

Do celów niniejszej pracy magisterskiej zostanie wykorzystany Apache ODE (*Apache Orchestration Director Engine*). Jest to narzędzie rozwijane przez społeczność otwartego oprogeamowania, zgodne ze standardem WS-BPEL. Silnik Apache ODE został zaimplementowany przy wykorzystaniu języka Java, jest on w rzeczywistości standardową aplikacją internetową uruchamianą na serwerach aplikacyjnych Java np. na serwerze Tomcat [8].

2.3.4. Podsumowanie

Należy pamiętać, że język BPEL nie jest jedynym słusznym rozwiązaniem służącym do implementacji procesów biznesowych. Bardzo często zdarza się, że projektanci systemów IT, którzy preferują

podejście top-down design (pl. projektowanie od góry do dołu), zaczynając projektować system IT przy pomocy notacji BPMN. Następnie zaś próbują przełożyć modele procesów biznesowych na kod wykonywalny w postaci BPEL. Nie jest to jednak słuszne podejście. Procesy BPMN zawierają często bardzo wiele szczegółów opisujących zachowanie samych operacji jednostkowych, natomiast pomijają szczegóły techniczne dotyczące usług sieciowych. Zazwyczaj okazuje się, że usługa sieciowa, która została by wygenerowana z diagramu BPMN w ogóle nie istnieje, a jej stworzenie może być mało sensowne.

Niestety nazwa BPEL jest w tym przypadku nie do końca trafna, ponieważ sugeruje, że BPEL jest językiem służącym do implementacji procesów biznesowych. W rzeczywistości jednak BPEL należy traktować jako narzędzie integracyjne. Narzędzie to ma służyć do definicji przebiegu komunikacji pomiędzy usługami sieciowymi, przy pomocy idei procesów biznesowych. Jest bardzo ważnym aby BPEL był stosowany właśnie w takich przypadkach, pozwoli to uniknąć poważnych błędów projektowych wynikających z zastosowania nieodpowiednich narzędzi.

2.4. Zestawienie technologii BPEL z aplikacjami mobilnymi

W rozdziałach poprzednich sporo uwagi zostało skupione na temacie modelowania procesów biznesowych. Został również przybliżony temat aplikacji mobilnych. Nadszedł czas aby odnieść się do tematu niniejszej pracy magisterskiej i spojrzeć na zestawienie tych dwóch technologii.

W postawionym problemie jak zwykle kluczowym jest odpowiednie podejście do tematu, uświadomienie sobie ograniczeń i odpowiednie dopasowanie technologii. Ograniczeń jest tutaj bardzo wiele głównie od strony aplikacji mobilnych.

Przykładów na wykorzystanie modelowania biznesowego w środowisku aplikacji mobilnych można wymyślić bardzo wiele, tym bardziej w przypadku aplikacji klienckich przeznaczonych dla biznesu. W tym ostatnim, środowisko mimo, że ukierunkowane na niewielkie aplikacje, jest pewnego rodzaju środowiskiem rozproszonym, w którym do tej pory procesy biznesowe sprawdzały się doskonale.

2.4.1. Adaptacja środowisk uruchomieniowych na platformach mobilnych

Jednym z możliwych sposobów podejścia do tematu może być próba bezpośredniej adaptacji środowisk uruchomieniowych dla procesów biznesowych na platformach mobilnych. Bezpośrednia adaptacja w tym przypadku oznacza próbę uruchomienia jakiegoś lekkiego środowiska uruchomieniowego na smartphonie. Po krótkim zastanowieniu można powiedzieć, że pomysł może być realny w realizacji - zasoby sprzętowe smartphonów są coraz większe, języki programowania wykorzystywane do tworzenia aplikacji mobilnych są identyczne jak języki do tworzenia środowisk uruchomieniowych (*Android* - język Java, *Windows Phone* - język `C#`). Gdy jednak kontynuujemy przemyślenia bardzo szybko natrafiamy na mur. Jak wspomniano w rozdziale 2.2, komunikacja aplikacji mobilnych odbywa się przede wszystkim metodą pull, głównie ze względu na zmienne warunki dostępu do mediów komunikacyjnych. W jaki sposób zatem środowisko uruchomieniowe miałby odbierać komunikację z zewnątrz? Jest przecież jeszcze metoda push, dostarczana przez dostawców platform mobilnych. Skorzystanie z niej jest

jednak złym pomysłem, ze względu na brak gwarancji dostarczenia komunikatu. Podejście adaptacyjne zatem nie jest dobrym pomysłem.

2.4.2. Wykorzystanie procesów biznesowych w systemach back end'owych

Jak wspomniano wyżej najwięcej zastosowań dla procesów biznesowych w kontekście aplikacji mobilnych można dostrzec w aplikacjach klienckich. Należy się zatem zastanowić w jaki sposób działają typowe aplikacje klienckie. Z powodu ograniczeń opisanych w poprzednim akapicie komunikacja zazwyczaj nie odbywa sie na zasadzie Peer-to-Peer, użytkownicy aplikacji klienckich jednak w jakiś sposób wymieniają między sobą informację. Jest to możliwe dzięki aplikacjom centralnym, zazwyczaj internetowym, zwanych systemami back'endowymi. Aplikacje mobilne w takich przypadkach komunikują się jedynie z systemem back end'owym nie posiadając informacji o sobie nawzajem. W tego rodzaju sposobie komunikacji można dostrzec bardzo wiele cech wspólnych z warstwą orkiestracji opisaną w rozdziale 2.3. Można zatem wyciągnąć wniosek, że procesy biznesowe w kontekście aplikacji mobilnych zostaną najefektywniej wykorzystane właśnie po stronie systemu back end'owego.

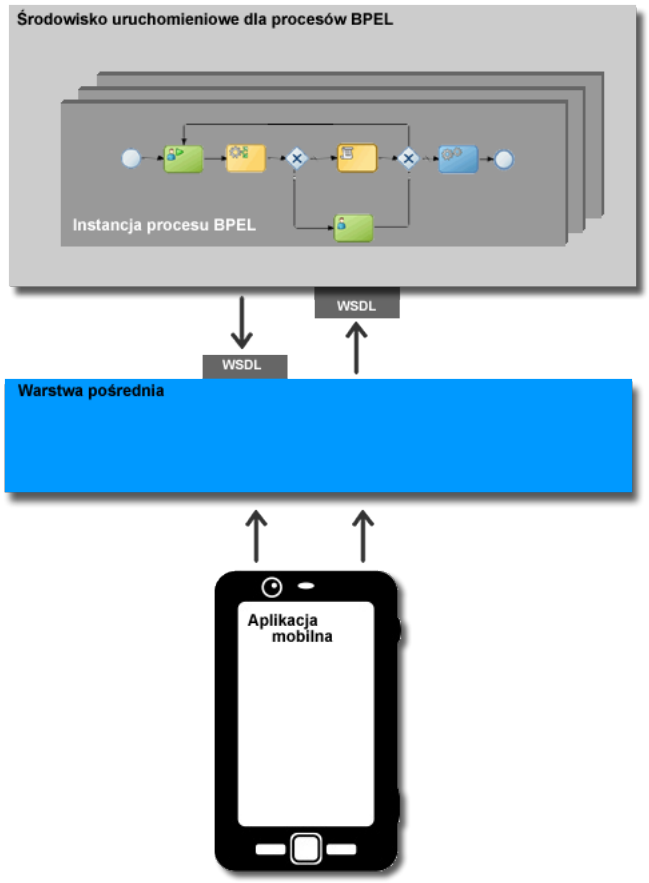
Podejście to można nazwać podejściem integracyjnym, w przypadku skorzystania z języka BPEL można sobie wyobrazić, że zrealizowany w ten sposób system back end'owy oprócz obsługi żądań aplikacji mobilnych mógłby komunikować się z innymi systemami przy wykorzystaniu usług sieciowych. Stworzony w ten sposób proces mógłby kontrolować przebieg komunikacji między aplikacjami mobilnymi traktując je jednocześnie na równi z innymi systemami. Niestety kolejny raz mimo sensownego pomysłu występują problemy w realizacji, mało tego ograniczenie kolejny raz jest takie samo. Proces BPEL jak opisano w rozdziale 2.3, do komunikacji poszczególnych węzłów wykorzystuje usługi sieciowe udostępniające odpowiednie interfejsy WSDL. Aplikacje mobilne nie są w stanie jednak udostępnić takich usług.

W tym przypadku istnieje sposób na poradzenie sobie z tym problemem, można stworzyć pewnego rodzaju adapter w postaci aplikacji internetowej, będącej warstwą pośrednią między procesami BPEL a aplikacjami mobilnymi. Taka warstwa pośrednia miałby za zadanie z jednej strony udostępnienie usług sieciowych dla procesów biznesowych, a z drugiej strony realizowała by komunikacje z aplikacjami mobilnymi, w celu przekazania żądań pochodzących od procesów biznesowych.

Opisanemu powyżej sposóbowi wykorzystania procesów biznesowych w aplikacjach mobilnych została poświęcona niniejsza praca, zostanie on dokładniej opisany w kolejnych rozdziałach.

3. Propozycja wykorzystania warstwy pośredniej

W rozdziale poprzednim podczas analizy sposobu działania klienckich aplikacji mobilnych, wyciągnięto wniosek, że najbardziej odpowiednim miejscem do wykorzystania procesów biznesowych, będą aplikacje back end'owe. Stwierdzono również, że w celu realizacji takiego rozwiązania konieczne jest zastosowanie dodatkowej warstwy pośredniej, realizującej komunikacje aplikacji mobilnych z procesami biznesowymi. Głównym powodem stosowania warstwy pośredniej jest brak możliwości udostępnienia przez aplikacje mobilne, usług sieciowych z którymi proces mógłby się komunikować.



Rysunek 3.1: Poglądowy schemat działania warstwy pośredniej

Na rysunku 3.1. przedstawiono poglądowy schemat działania opisanego powyżej rozwiązania. Diagram ma na celu lepsze zobrazowanie problemu i wyłonienie funkcjonalności, które warstwa pośrednia musi realizować. Na schemacie można dostrzec trzy rodzaje systemów: środowisko uruchomieniowe dla procesów biznesowych, warstwę pośrednią oraz aplikację mobilną. Strzałki umieszczone na schemacie przedstawiają kierunki komunikacji pomiędzy poszczególnymi systemami. Komunikacja między aplikacją mobilną a warstwą pośrednią zawsze odbywa się z inicjatywy aplikacji mobilnej, jest to związane ze specyfiką dostępu aplikacji do mediów komunikacyjnych. Wynika z tego, że jedną z funkcjonalności które musi realizować warstwa pośrednia jest buforowanie żądań pochodzących z procesów biznesowych do momentu aż aplikacja mobilna się po nie zgłosi sama.

Aplikacje mobilne są specyficzne pod jeszcze jednym względem - poszczególne instancje aplikacji uruchamiane są zawsze na urządzeniach personalnych. Bardzo mocny akcent zatem kładziony jest tutaj na samych użytkowników. Gdy przyglądniemy się procesowi BPEL to możemy dostrzec, że wątek ludzki jest w nich zupełnie pomijany. Proces komunikuje się z usługą sieciową jedynie w celu rozwiązania jakiegoś konkretnego zadania i zazwyczaj oczekuje na rezultat.

3.1. BPEL4People

Jak wspomniano powyżej procesy WS-BPEL traktują użytkowników po macoszemu, skupiają się przede wszystkim na sekwencji wykonywania kolejnych operacji. Problem dostrzegli projektanci z zespołów IBM oraz SAP, a proponowaną przez nich odpowiedzią jest powstanie rozszerzenia do BPEL nazwanego *BPEL4People*.

Twórcy standardu argumentują powstanie kolejnej specyfikacji koniecznością uwzględnienia aspektu ludzkiego w procesach BPEL. Podają szereg przykładów, w których interakcja procesu z człowiekiem jest konieczna. Jednym z przykładów jest konieczność dokonywania decyzji akceptacji lub odrzucenia wyniku jakiejś operacji. Innym przykładem jest wprowadzanie dodatkowych danych niezbędnych do kontynuacji wykonania procesu, za pomocą formularzy[9, str. 4].

Specyfikacja opiewa w szereg bardzo ciekawych pomysłów integracji interakcji użytkowników z procesami BPEL. Porusza problem przejścia pomiędzy serwisami dostosowanymi dla ludzi a serwisami zautomatyzowanymi, zwraca uwagę na problem monitorowania zadań przeznaczonych dla ludzi, opisuje problem terminów ostatecznych wykonania zadań oraz eskalacji w przypadku ich przekroczenia[9, str. 6].

3.1.1. WS-HumanTask

Bezpośrednio powiązany z rozszerzeniem *BPEL4people* jest temat usług sieciowych dla ludzi, które są wykorzystywane przez to rozszerzenie.

WS-HumanTask są usługami sieciowymi 'implementowanymi' przez ludzi. Pozwalają na integrację ludzi z aplikacjami zorientowanymi na usługi (SOA). Specyfikacja udostępnia dwa interfejsy do komunikacjami z zadaniami. Pierwszy z nich jest przeznaczony dla automatów czyli aplikacji rozproszonych,

drugi natomiast jest przeznaczony dla ludzi i pozwala na operacje zadaniami. Zadania opisane w specyfikacji posiadają przypisania do konkretnych grup docelowych. Przypisania te definiują kto powinien być uprawniony do operacji na zadaniach. Specyfikacja definiuje również meta dane dotyczące zadań, pozwalające na określenie w jaki sposób zadania mają być prezentowane na różnych urządzeniach. Zadanie mogą również definiować reakcje na terminy ostateczne ich wykonania, w postaci mechanizmu eskalacji.

3.1.2. Implementacje

Z punku widzenia proponowanego w niniejszej pracy magisterskiej rozwiązanie zastosowanie *BPEL4People* rozwiązuje szereg problemów związanych ze spersonalizowaną specyfiką aplikacji mobilnych. Kolejne instancje aplikacji mobilnych można utożsami z użytkownikami. Warstwa pośrednia udostępniałaby w takim podejściu WS-HT, z którymi komunikowałby się proces *BPEL4People*. Niestety rzeczywistość w tym przypadku okazuje się brutalna. Rozszerzenie do BPEL jeśli nie zostaje zaimplementowane jest bezwartościowe. W przypadku *BPEL4People* mimo tego że standard istnieje już 9 lat, nie został jeszcze wdrożony w żadnym z popularnych środowisk uruchomieniowych. Warstwa pośrednia powinna więc, samodzielnie uwzględniać aspekt ludzki pomijany przez procesy WS-BPEL.

Mimo braku implementacji można dostrzec pewne korzyści płynące z powstania rozszerzenia *BPEL4People*. Są nimi z pewnością pomysły dotyczące rozwiązania problemu dystrybucji żądań pochodzących z procesów WS-BPEL między różnymi użytkownikami. W niniejszej pracy magisterskiej niejednokrotnie skorzystamy z tej wiedzy.

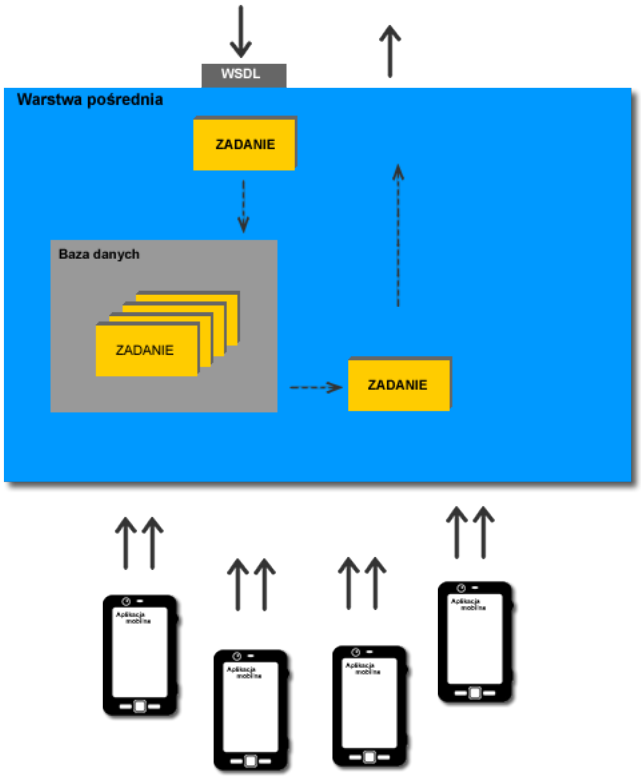
3.2. Zadania

Pierwszym z pomysłów zaczerpniętych z *BPEL4People* jest pomysł na traktowanie kolejnych żądań pochodzących z procesów WS-BPEL jako zadań, a aplikacji mobilnych jako zespołu specjalistów potrafiących rozwiązać te zadania. Podejście to rozwiązuje problem konieczności buforowania żądań procesu ze względu na komunikacje pull od strony aplikacji mobilnej.

Na rysunku 3.2. przedstawiono poglądowy schemat wykorzystania zadań w warstwie pośredniej. Strzałkami przerywanymi został oznaczony podstawowy cykl życia zadania. Po otrzymaniu żądanie od procesu BPEL warstwa pośrednia utworzy nowe zadanie i zapisze go w bazie danych. Aplikacje mobilne korzystając z lekkiego interfejsu komunikacyjnego, będą potrafiły pobierać zadania z bazy danych w celu ich rozwiązania. Po rozwiązaniu zadania, następować będzie wysłanie odpowiedzi do procesu biznesowego.

3.2.1. Dystrybucja zadań

Przedstawiony powyżej schemat nie jest jednak kompletny. Oczywistym jest, że aby proces biznesowy miał sens powinien wykonywać więcej niż jedną operację, dla każdej operacji w proponowanym

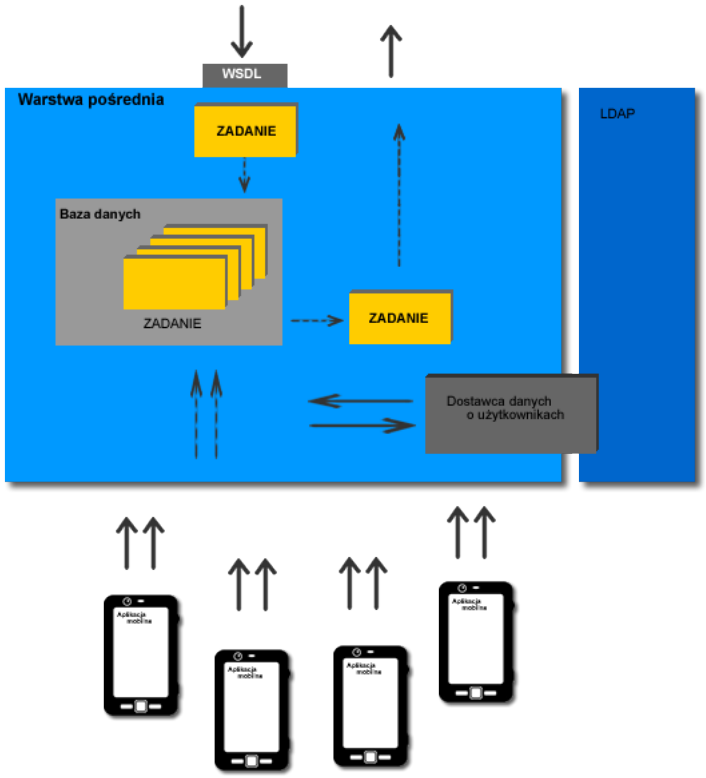


Rysunek 3.2: Poglądowy schemat wykorzystania zadań w warstwie pośredniej.

rozwiązaniu istnieć będzie zupełnie inna klasa zadań. Nie trudno dojść do wniosku, że każda z tych klas obsługiwana może być przez zupełnie różne grupy użytkowników. Na przykład niemal w każdym przedsiębiorstwie zadania dyrektora są kompletnie inne od zadań zwykłego pracownika, ale są one między sobą w jakiś sposób powiązane na przykład występują w tym samym procesie. Wnioskiem z tego płynącym jest konieczność wprowadzenia w warstwie pośredniej mechanizmu przydzielania zadań poszczególnym użytkownikom. W przypadku tym ponownie rozszerzenie *BPEL4People* przewidziało ten problem i zaproponowało rozwiązanie. Wprowadza ono do każdej klasy zadań, definicje grupy docelowych adresatów.

Na rysunku 3.3 przedstawiono schemat warstwy pośredniej uzupełniony o zarządzanie użytkownikami. Aplikacje mobilne podczas rozwiązywania zadań będą przedstawiać się przy pomocy nazwy użytkownika. Przed pobraniem zadań z bazy danych dostawca danych o użytkownikach dostarczy informacji niezbędnych do określenia zadań specyficznych dla danego użytkownika. Sposób ten zapewni, że zadania należące do dyrektora przedsiębiorstwa nie trafią do zwykłych pracowników.

Zastosowanie dodatkowej warstwy nazwanej *Dostawcą danych o użytkownikach* jest tutaj celowe i będzie wymagać osobnej implementacji dla różnych rodzajów sposobów przechowywania danych o użytkownikach. Użytkownicy nie koniecznie muszą pochodzić z systemu katalogowego takiego jak LDAP. Możliwości przechowywania danych o użytkownikach jest bardzo wiele, dla prostych systemów lista użytkowników może być nawet predefiniowana.



Rysunek 3.3: Schemat warstwy pośredniej uzupełniony o zarządzanie użytkownikami.

3.2.1.1. Context aware middleware

Dodatkowym atutem udostępnienia możliwości implementacji dostawcy jest możliwość integracji adaptera z systemami zwanymi *Context aware middleware*.

Context aware middleware jest oprogramowaniem które za pomocą danych dostarczanych z możliwie jak największej ilości źródł, stara się rozwiązać zadany problem. Rozwiązanie to bardzo szerokie zastosowanie znajduje głównie w aplikacjach mobilnych, dlatego że kontekst w przypadku tych aplikacji zmienia się bardzo dynamicznie. Architektura skupia się przede wszystkim na dostarczeniu informacji opisujących takie wymiary jak: - miejsce, Gdzie ? - czas, Kiedy? - stan np. Jaka jest temperatura otoczenia? - podmiot, Kto? [10]

Bazując na tych danych stara się rozwiązywać dany problem w najbardziej optymalny sposób, na przykład przydzielając zadania osobie która jest najbliżej, która jest najmniej obciążona, która jest aktualnie w pracy itd.

Aby zastosować *Context aware middleware* w warstwie pośredniej wystarczy odpowiednia implementacja dostawcy danych o użytkownikach. Uzyskamy dzięki takiemu rozwiązaniu potężne narzędzie potrafiące dystrybuować zadania w bardzo optymalny sposób zapewniając ich szybką realizację.

3.2.2. REST API

Interfejs komunikacji warstwy pośredniej od strony procesów biznesowych został wymuszony przez specyfikację BPEL. Aby skompletować projekt warstwy pośredniej konieczne jest zdefiniowanie interfejsu komunikacyjnego od strony aplikacji mobilnych.

Interfejs ten powinien przede wszystkim rozwiązywać podstawowy problem, występujący podczas rozwiązywania zadań przez wielu użytkowników jakim są konflikty. Nie może on dopuścić do sytuacji w której jedno zadanie byłoby rozwiązywanie przez wielu użytkowników jednocześnie. W celu rozwiązanie tego problemu konieczne jest wprowadzenie statusów zadań. W proponowanej warstwie pośredniej będą występowały następujące statusy zadania:

– gotowe do wykonania (ang. *ready*) – zadanie zostanie oznaczone tym statusem tuż po utworzeniu. Użytkownik aplikacji mobilnych szukając zadań do wykonania otrzyma listę zadań oznaczonych właśnie tym statusem.

– przypisane (ang. *climed*) – status ten oznacza, że któryś z użytkowników zadeklarował chęć jego wykonania. Będzie ono od tej pory dostępne tylko dla tego użytkownika. Pozostali użytkownicy poszukujący zadań do wykonania go nie zobaczą.

– rozwiązane (ang. *completed*) – statusem tym oznaczone będą zadania zrealizowane z sukcesem, których rezultat został odesłany do procesu biznesowego.

– błąd podczas rozwiązywania (ang. *failed*) – statusem tym oznaczone zostaną wszystkie zadania, podczas których rozwiązania wystąpił problem techniczny.

Znając poszczególne statusy nadszedł czas na określenie interfejsu komunikacyjnego z aplikacjami mobilnymi. Ponieważ wśród aplikacji mobilnych najbardziej popularnym medium komunikacyjnym jest połączenie sieciowe na pewno najrozsądniejszym pomysłem będzie z niego skorzystanie. Jak wspomniano w rozdziale 2.2, podczas projektowania aplikacji mobilnych należy zwrócić uwagę na to aby wybierane technologie nie obciążały zbytnio ograniczonego pod względem zasobów urządzenia mobilnego. Dobrym wyborem w takim wypadku będzie REST.

3.2.2.1. Usługi sieciowe RESTFul

REST (ang. *Representational State Transfer*) jest wzorcem narzucającym dobre praktyki tworzenia architektury aplikacji rozproszonych. Usługi sieciowe RESTFul są usługami zaimplementowanymi na bazie protokołu HTTP i głównych zasad REST [11]. Podstawą architektury REST jest zasób, zasób jest tutaj traktowany nie tylko jako jakiś byt, który można pobrać z aplikacji internetowej za pomocą protokołu HTTP. Zasobem w architekturze REST są również usługi udostępniane przez aplikacje internetowe.

Kolejnym z głównych założeń architektury REST jest wykorzystanie metod udostępnianych przez protokół HTTP do definicji operacji wykonywanych na danym zasobie. Protokół HTTP oprócz najbardziej popularnych metod jak GET i POST udostępnia również całą gamę pozostałych metod. W architekturze REST oprócz tych dwóch najpopularniejszych bardzo często wykorzystywane są również PUT

oraz DELETE. Znaczenie poszczególnych metod w kontekście operacji na zasobie może przedstawiać się następująco:

– GET – pobranie danych

– POST – dodawanie danych

– PUT – edycja danych

– DELETE – usuwanie danych

3.2.2.2. Definicja interfejsu

Warstwa pośrednia do komunikacji z aplikacjami mobilnymi będzie udostępniać następujące funkcje:

– lista zadań do wykonania – metoda nie będzie przyjmować żadnych parametrów dodatkowych, kontekst użytkownika zostanie określony na podstawie nagłówków HTTP wymaganych przez *HTTP Basic Authentication*.

– przypisz do zadania – metoda jako parametr będzie przyjmować unikalny identyfikator zadania

– zrezygnuj z zadania – metoda jako parametr będzie przyjmować unikalny identyfikator zadania

– rozwiąż zadanie – metoda jako parametr będzie przyjmować unikalny identyfikator zadania oraz wynik rozwiązania zadania.

3.3. Wybrane technologie

Niemal w każdym projekcie informatycznym kluczową sprawą jest dobór odpowiednich technologii. Do realizacji warstwy pośredniej potrzebna będzie technologia webowa potrafiąca udostępnić usługi sieciowe, potrafiąca komunikować się z bazą danych i posiadająca wsparcie dla technologii REST. Niemal wszystkie popularne języki programowania spełniają każdy z tych warunków. Należy zatem pójść o jeden krok dalej i poszukać gotowego frameworka (zestawu gotowych bibliotek i wzorców postępowania), który przy wykorzystaniu jednego z tych języków będzie potrafił zrealizować przynajmniej cześć zadań. Wybór w niniejszej pracy magisterskiej padł na Spring Framework zaimplementowany w języku Java.

3.3.1. Spring Framework

Spring Framework zapewnia kompleksowy model programistyczny i konfiguracyjny dla nowoczesnych aplikacji biznesowych opartych na języku Java. Kluczowym elementem jest wsparcie infrastrukturalne na poziomie aplikacji: Spring skupia się na 'hydraulice' aplikacji dla przedsiębiorstw, tak, że zespoły mogą skupić się na logice biznesowej na poziomie aplikacji, bez konieczności określania środowisk programowania [12].

– Spring Web Services – w przypadku warstwy pośredniej jedną z głównych funkcjonalności koniecznych do realizacji jest udostępnienie usług sieciowych do komunikacji z procesami biznesowymi. Spring Framework posiada specjalnie do tego celu przygotowany moduł nazwany Spring Web Services. Moduł ten koncentruje się tworzeniu usług sieciowych na podstawie dokumentów SOAP przy wykorzystaniu stylu, umowa-najpierw (ang. *contract-first*). Styl ten polega na definicji interfejsów w postaci plików WSDL, przed stworzeniem konkretnej implementacji usługi [13].

– Spring MVC – moduł MVC jak sama nazwa wskazuje jest przeznaczony do tworzenia aplikacji opartych o jeden z najbardziej popularnych wzorców projektowych *Model View Controller*. Spring MVC jest przeznaczony przede wszystkim do zastosowania wzorca w aplikacjach webowych gdzie do komunikacji wykorzystywany jest protokół HTTP. W przypadku aplikacji webowych widokiem zazwyczaj jest dynamicznie generowany kod HML, nie jest to jednak reguła. W przypadku warstwy pośredniej MVC zostanie wykorzystane do przygotowania REST API. Widokiem w takim wypadku będą dane opisane za pomocą dokumentów JSON [14].

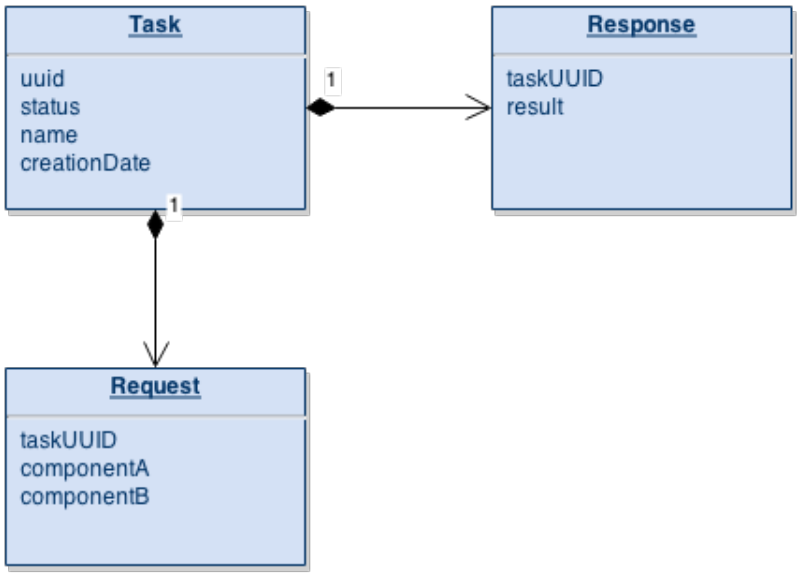
– Spring Data – Oprócz wyżej wymienionych modułów, Spring Framework dostarcza również wsparcie do obsługi komunikacji z bazą danych, najczęściej do tego celu wykorzystując popularne ORM'y (Biblioteki do mapowania obiektowo relacyjnego) jak Hibernate. W przypadku warstwy pośredniej istnieje konieczność przechowywania zadań w bazie danych dlatego Spring Data również zostanie wykorzystany.

– Spring Secutiry – W niniejszym rozwiązaniu uwierzytelnienia użytkowników REST API będzie odbywało się za pomocą *HTTP Basic Authentication*. Spring posiada również dodatkowe wsparcie do obsługi bezpieczeństwa przy pomocy *HTTP Basic Authentication*, wsparcie to udostępnione jest za pomocą modułu Spring Security. Oprócz prostej autoryzacji Spring Security posiada również mechanizmy do zapamiętywania sesji użytkownika, zarządzanie jego uprawnieniami itd. W niniejszym rozwiązaniu nie będziemy z nich jednak korzystać.

3.3.2. MongoDB

Bardzo ważną decyzją jest również wybór odpowiedniej bazy danych. Wymagania dotyczące warstwy pośredniej jasno określają, że głównym modelem przechowywanym w bazie danych będzie zadanie. Zadanie oprócz podstawowych informacji takich jak identyfikator, status, data utworzenia itd. będzie posiadał wszelkie informacje potrzebne do jego utworzenia, oraz jeśli zadanie zostanie wykonane posiadać będzie również informacje o rozwiązaniu. W przypadku skorzystania z relacyjnej bazy danych klaruje nam się przykładowy schemat bazy danych przedstawiony na rysunku 3.4.

W przypadku pojedynczego zadania schemat może okazać się słuszny, problem powstaje gdy warstwa pośrednia obsługuje więcej niż jedno zadanie. W takim przypadku każdy kolejny rodzaj danych wejściowych poszczególnych zadań powinien znajdować się w osobnej tabeli. Mało tego dane wejściowe zadań nie muszą ograniczać się jedynie do płaskiej struktury danych. Mogą one zawierać argumenty



Rysunek 3.4: Diagram ERD przykładowego schematu bazy danych relacyjnej.

będące osobnymi strukturami danych. W takich przypadkach, z punktu widzenia normalizacji bazy danych, każda z tych struktur powinna również znajdować się w osobnych tabelach. Taka postać rzeczy ma bardzo negatywny wpływ na poziom komplikacji zapytań wyciągających zadania dla poszczególnych użytkowników. Mogą istnieć przypadki w których na podstawie danych wejściowych będzie potrzeba ograniczenia adresatów zadania, np. wybrania użytkowników w obrębie kilku kilometrów od adresu znajdującego się w danych wejściowych.

Sposobem na rozwiązanie problemu nie koniecznie płaskiej struktury danych wejściowych może być zastosowanie typu danych xml. W takich przypadkach dane wejściowe zostałyby serializowane do dokumentów xml i w takiej postaci zapisane w bazie danych. Kolejny raz problemem są jednak skomplikowane zapytania do bazy danych.

Odpowiedzią na postawione wymagania są dokumentowe bazy danych - w przypadku niniejszej pracy magisterskiej będzie to baza danych MongoDB [15]. Przechowuje ona dane w postaci kolekcji dokumentów. Dokumenty przechowywane w bazie danych mogą posiadać dowolnie skomplikowaną strukturę danych, a odpowiednio przygotowane zapytania będą potrafiły filtrować listę tych dokumentów na podstawie wszystkich argumentów znajdujących się w tej strukturze.

Przykładowy dokument bazy danych MongoDB może przedstawiać się następująco:

```
{
  uuid : '6942cd9b-3b3b-49bd-9b18-de68384bd22a',
  name : 'createWarehouseDocument',
  state : '',
  createDate : '2014-07-24',
  priority: 3,
  request : {
    document :{
```

```
      type: 'inbound',
      company : {
        name : 'AGH',
        address : {
          city: 'Krakow',
          streat : 'al. A. Mickiewicza',
          number: '30',
          country: 'Polska'
        }
      },
      positions: [...]
    }
  }
}
```

Listing 3.1: Zadanie w postaci dokumentu MongoDB.

Przedstawiony wyżej dokument jest przykładem zadania którego celem jest stworzenie odpowiedniego dokumentu magazynowego. Możemy sobie wyobrazić sytuację, że dokumenty dla kontrahentów z poszczególnych krajów są tworzone przez różnych użytkowników. Przykładowe zapytanie znajdujące wszystkie zadania dla użytkownika obsługującego Polskę wyglądało by następująco.

```
{'request.document.company.address.country': 'Polska'}
```

Listing 3.2: Zapytanie MongoDB.

W przypadku relacyjnej bazy danych powyższa struktura wymagała by stworzenie sześciu tabel a zapytanie które dałby by dokładnie taki sam rezultat wyglądało by następująco:

```
SELECT * FROM Tasks t
  JOIN Request r ON t.id = r.task_id
  JOIN Document d ON r.id = d.request_id
  JOIN Company c ON d.id = c.document_id
  JOIN Address a ON c.id = a.company_id
WHERE a.country = 'Polska';
```

Listing 3.3: Zapytanie SQL.

3.3.3. Maven

Kolejna technologią wykorzystaną do realizacji idei przestawionej w niniejszej pracy magisterskiej jest Maven. Wykorzystanie tej technologii wynika z decyzji wykorzystania języka Java. Maven jest narzędziem wspomagającym budowanie projektów stworzonych w języku Java. Odpowiada on za skompletowanie wszystkich wymaganych bibliotek zdefiniowanych w odpowiednim pliku konfiguracyjnym - *pom.xml*. Biblioteki te pobierane są ze specjalnie przygotowanych do tego celu repozytoriów dostępnych w sieci, na lokalną maszynę. Biblioteki te są następnie wykorzystywane w procesie budowania plików wynikowych [16].

3.4. Sprawienie rozwiązania generycznym

Przedstawiony do tej pory model warstwy pośredniej skupia się przede wszystkim na problemie komunikacji pomiędzy procesem biznesowym oraz aplikacją mobilną, opisuje również sposób przechowywania i dystrybucji zadań. Model jest przy okazji bardzo ogólny i możliwy do zastosowania bez względu, na rozwiązywany problem. Nie ma w nim mowy o konkretnych przypadkach zastosowań.

Dobrym pomysłem będzie więc stworzenie rozwiązania na tyle elastycznego aby mogło być wykorzystywane bez względu na rozwiązywany problem. W środowisku IT taki model postępowanie nazywany jest generycznym. W przypadku warstwy pośredniej sposób zarządzania zadaniami jest taki sam dla wszystkich rodzajów zadań - utworzenie zadania po otrzymaniu żądania od procesu, udostępnienie zadania aplikacją mobilnym, odpowiedź do procesu. Główną różnicą są dane wejściowe różnych żądań oraz sposób dystrybucji zadań pomiędzy aplikacjami na podstawie tych danych. Problem różnych struktur danych wejściowych dla różnych zadań rozwiązuje nam wybór bazy danych MognoDB, który będzie potrafił zapisać te struktury bez dodatkowych konfiguracji, oraz będzie potrafił je przeszukiwać za pomocą prostych zapytań.

Problem różnego sposobu dystrybucji danych rozwiążemy za pomocą pliku konfiguracyjnego, który będzie opisywał poszczególne zadania, oraz definiował jakie grupy użytkowników są uprawnione do ich rozwiązywania.

Opisany model można zobrazować za pomocą schematu widocznego na rysunku 3.5.

Podsumowując, opisane w niniejszej pracy magisterskiej rozwiązanie będzie generyczną biblioteka służącą do tworzenie aplikacji webowej. Aplikacja ta będzie warstwa pośrednią pomiędzy procesem biznesowym a aplikacjami mobilnymi.

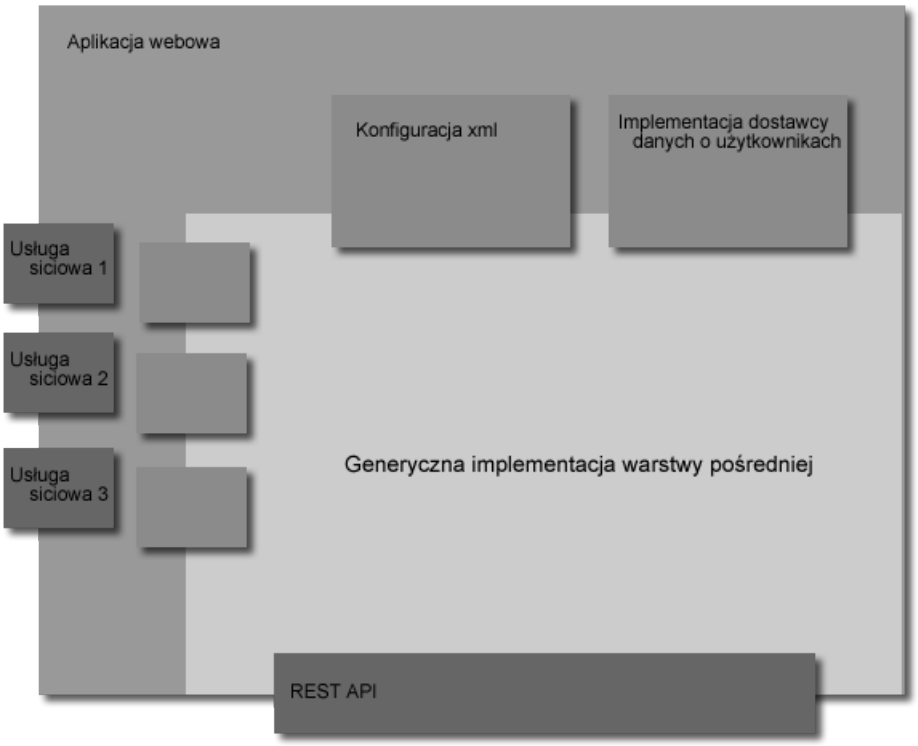
3.5. Implementacja

Implementacja biblioteki zostanie przedstawiona w sposób schematyczny. Zostanie przedstawiony projekt klas występujących w rozwiązaniu wraz z ich opisem. W następnej część zostaną przedstawione diagramy sekwencji najważniejszych funkcjonalności rozwiązania, opisujące przebieg wywołań kolejnych funkcji zdefiniowanych klas.

3.5.1. Diagram klas

Diagram klas przedstawiony na rysunku 3.6. przedstawia zestaw klas, które wymagają dodatkowego objaśnienia:

– *TaskService* – jest to klasa odpowiedzialna za obsługę żądań pochodzących z procesów biznesowych. Każdy rodzaj zadania powinien posiadać własną instancję tej klasy, która z kolei powinna być wykorzystywana przez usługę sieciową odpowiedzialną za jego obsługę. Klasa posiada dwie metody, *setConfig* odpowiedzialną za ustawienie podstawowych informacji o zadaniu, które będzie obsługiwać instancja tej klasy, takich jak nazwa, lista klas definiujących dane wejściowe



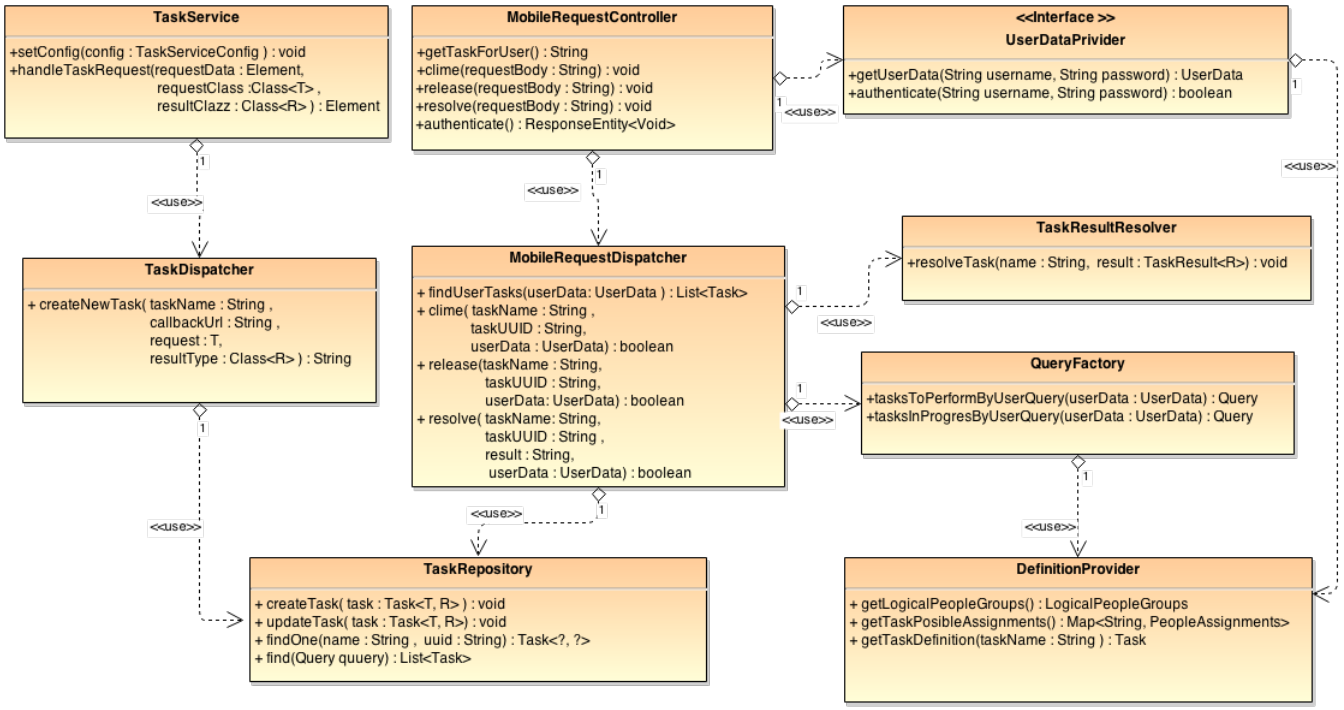
Rysunek 3.5: Poglądowy schemat wydzielenia biblioteki odpowiedzialnej za obsługę warstwy pośredniej.

i wyjściowe zadania. Kolejną metodą jest metoda *handleTaskRequest* metoda ta przyjmuje jako parametr obiekt typu *Element* zawierający dane wejściowe w postaci xml. Metoda przetwarza te dane na odpowiednie klasy i zapisuje zadanie w bazie danych do późniejszej realizacji. Zwraca ona obiekt typu *Element* zawierający informacje o unikalnym identyfikatorze przypisanym zadaniu, obiekt ten jest plikiem xml który powinna zwrócić usługa sieciowa. Metoda *handleTaskRequest* powinna zostać wywołana przez usługę sieciową odbierającą żądanie od procesu biznesowego.

– *TaskDispatcher* – Klasa odpowiedzialna za przetwarzanie, przyjętych przez klasę *TaskService* zadań. Posiada ona jedną metoda służącą do zapisu zadania przetworzonego przez *TaskService*, nazwaną *createNewTask*.

– *TaskRepository* – repozytorium zadań odpowiedzialne za obsługę bazy danych. Posiada metody służące do zapisu i edycji zadań oraz do wyszukiwania zarówno pojedynczych zadań, jak i ich listy.

– *MobileRequestController* – klasa przeznaczona do udostępnienia REST API. Posiada ona metody udostępniane przez interfejs przeznaczony do aplikacji mobilnych. Klasa ta korzystając ze Spring Security, odnajduje kontekst aktualnego użytkownika, pobiera informacje o nim za pomocą *UserDataProvider*'a przekazując następnie kontrole *MobileRequestPrivider*'owi.



Rysunek 3.6: Diagram klas implementacji warstwy pośredniej.

– *UserDataProvider* – interfejs służący do określenia metod które musi implementować dostawca informacji o użytkowniku. Implementacje *UserDataProvider*'a będą wykorzystywana między innymi przez Spring Security.

– *DefinitionProvider* – klasa to odpowiedzialna jest za deserializację pliku xml zawierającego konfiguracje poszczególnych zadań, oraz dostarczenie tej konfiguracji do pozostałych klas warstwy pośredniej.

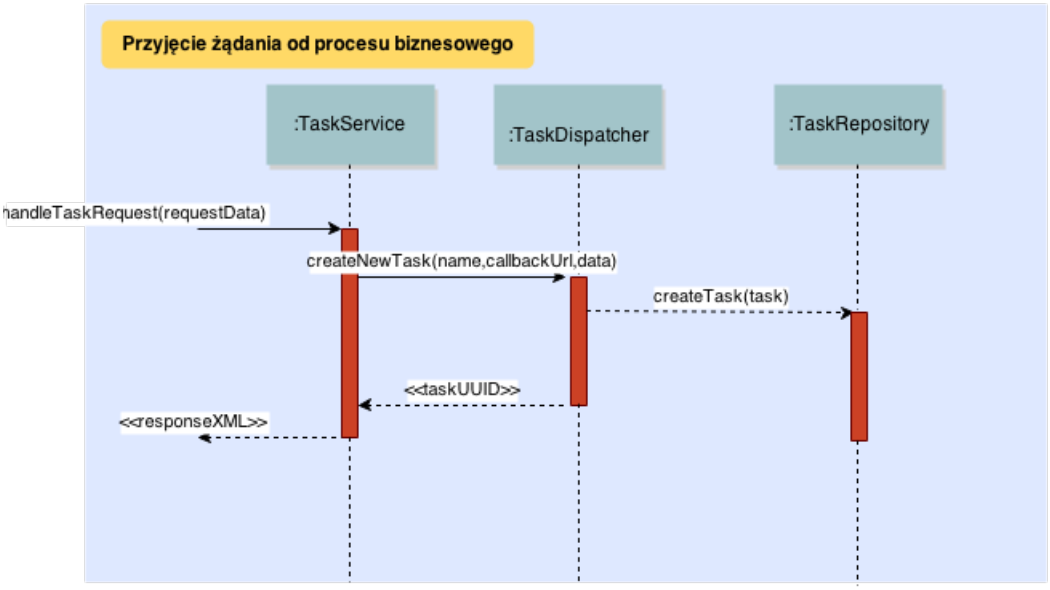
– *MobileRequestDispatcher* – klasa odpowiedzialna za obsługę żądaniami pochodzącymi z aplikacji mobilnych. Współpracuje z klasami przeznaczonymi do generowania zapytań, wyciągania danych z bazy oraz odpowiedzialnymi za odesłanie rozwiązania do procesów biznesowych.

– *QueryFactory* – głównym zadaniem tej klasy jest przygotowanie zapytań pozwalających odnaleźć zadania dostępne dla danego użytkownika. Zapytania generowane są na podstawie konfiguracji xml oraz danych o użytkowniku.

– *TaskResultResolver* – klasa posiada tylko jedną metodę nazwaną *resolveTask* służąca do wysłania odpowiedzi do procesu biznesowego zawierającej rezultat wykonania zadania.

3.5.2. Diagramy sekwencji

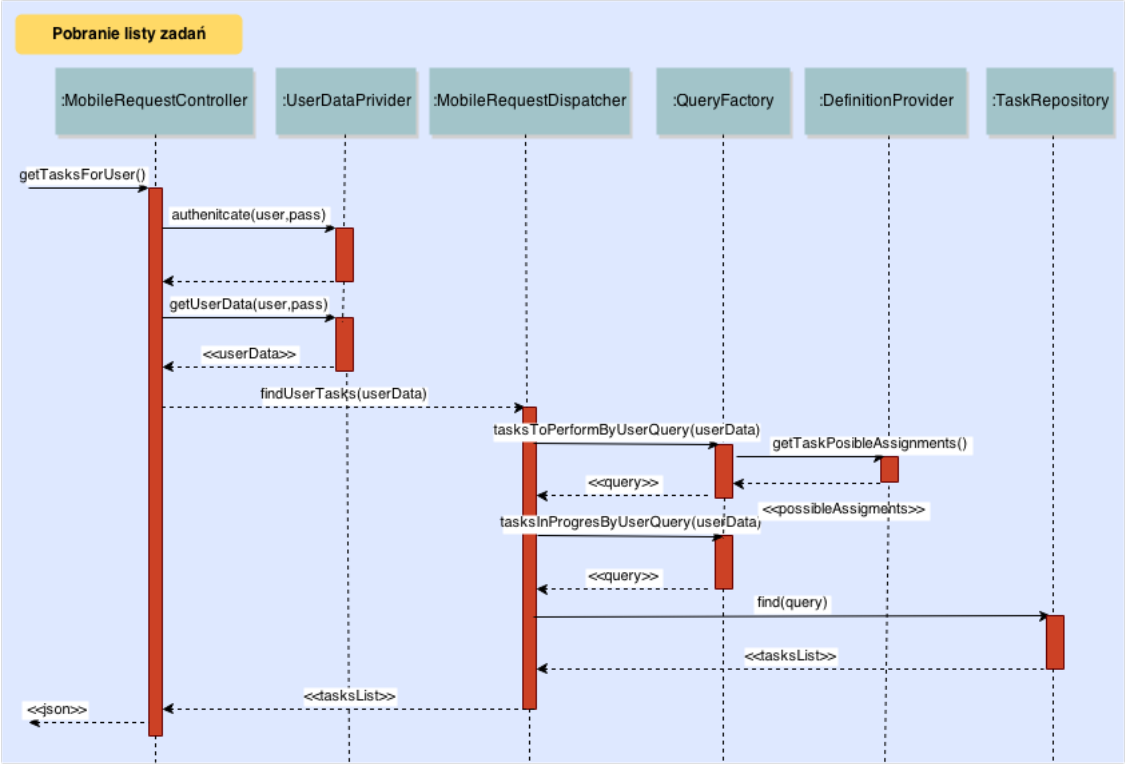
Sposób działania powyżej opisanych klas zostanie przedstawiony za pomocą diagramów sekwencji. Diagramy przedstawiają sekwencje kolejnych wywołań funkcji w celu realizacji funkcjonalności dostarczanych przez warstwę pośrednią.



Rysunek 3.7: Diagram sekwencji obsługi żądania pochodzącego z procesu BPEL.

Na rysunku 3.7 zobaczyć można diagram sekwencji funkcjonalności - przyjęcie żądania od procesu biznesowego. Usługa sieciowa korzystając z konkretnej instancji klasy *TaskService* uruchamia metodę *handleTaskRequest* przekazując do niej obiekt zawierający dokument xml z danymi wejściowymi. Wewnątrz metody, dokument xml zostaje zdeserializowany do postaci obiektów podanych w konfiguracji klasy *TaskService*. Zostaje również wydobyty adres url usługi przyjmującej ostateczny rezultat zadania. Sterowanie przekazywane jest następnie do klasy *TaskDispatcher* w celu utworzenie nowego zadania w bazie danych. Metoda *createNewTask* klasy *TaskDispatcher* oprócz stworzenia nowego dokumentu w kolekcji w bazie danych MongoDB za pomocą repozytorium *TaskRepository*, odpowiedzialna jest za generację unikalnego identyfikatora w postaci klucza GUID. Unikalny identyfikator jest następnie zwracany ponownie do *TaskService*'u gdzie zostaje opakowany w dokument xml, który zostanie zwrócony do procesu biznesowego. Wysłanie unikalnego klucza do procesu jest w tym przypadku krytyczne, po stronie silnika uruchomieniowego procesów biznesowych BPEL będzie istniało bardzo wiele instancji procesów w różnym stadium wykonania. Komunikacja procesu z warstwą pośrednią odbywa się w sposób asynchroniczny, poprzez kolejne wywołania aktywności request (wysłanie żądania), by następnie przejść do aktywności recieve (oczekiwać na jego rezultat). W między czasie proces może również wykonać inne operacje. Aby skojarzyć aktywność wysłania żądania z otrzymaniem żądania w przypadku wielu instancji procesu konieczna jest identyfikacja tych żądań za pomocą tego samego unikalnego klucza. W procesach biznesowych BPEL mechanizm ten został nazwanych mechanizmem korelacji. Aby

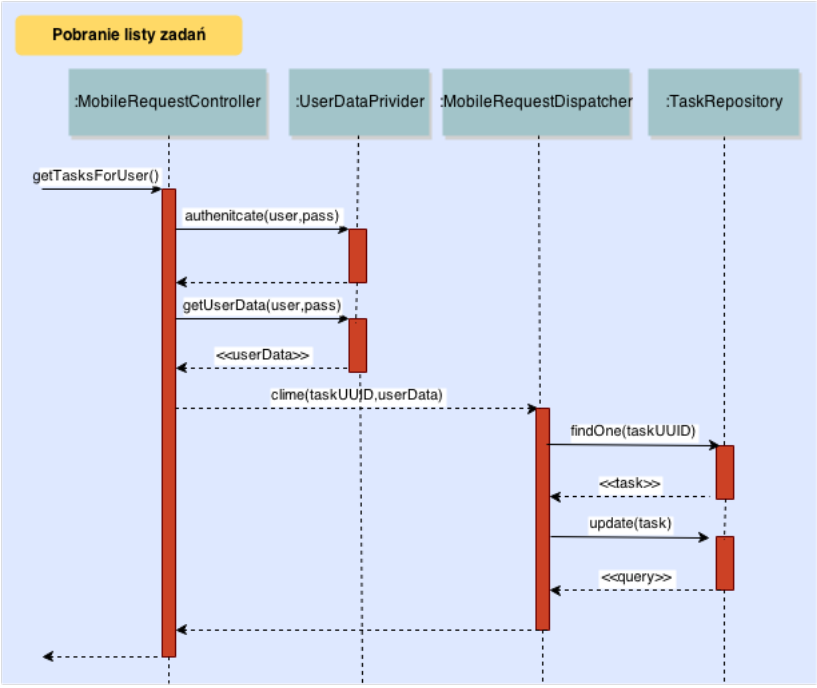
skorzystać zatem z mechanizmy korelacji, podczas tworzenia nowego zadania proces biznesowy musi otrzymać unikalny klucz tego zadania by następnie podczas otrzymania rezultatu zadania był w stanie skojarzyć ten rezultat z konkretną instancją procesu. Rezultat w takim wypadku również musi zostać podpisany unikalnym identyfikatorem zadania.



Rysunek 3.8: Diagram sekwencji pobrania listy zadań dostępnych dla użytkownika.

Rysunek 3.8 przedstawia diagram sekwencji pobierania listy zadań przez aplikację mobilną. Sekwencja w tym przypadku zaczyna się od klasy *MobileRequestController*, która jest odpowiedzialna za obsługę żądań odpowiednich żądań http. Kontroler ten rozpoczyna wykonanie metody od autentykacji użytkownika którym przedstawia się żądanie HTTP (z wykorzystaniem *HTTP basic authentication*). Po poprawnej autentykacji zostają pobrane informacje o aktualnym użytkowniku, informacje te zostaną wykorzystane do wygenerowania listy zadań dla niego dostępnych. Sterowanie zostaje następnie przekazane do klasy *MobileRequestDispatcher*, która za pomocą *QueryFactory* zbuduje dwa zapytania, jedno do wyciągnięcia nowych zadań do których użytkownika ma dostęp. Drugie zapytanie pokryje wszystkie zadania w trakcie realizacji przypisane do użytkownika. Zapytania te posłużą następnie do wyciągnięcia listy zadań z bazy danych przy pomocy repozytorium *TaskRepository*. Ostatnim krokiem będzie serializacja wydobytych zadań do postaci dokumentu JSON.

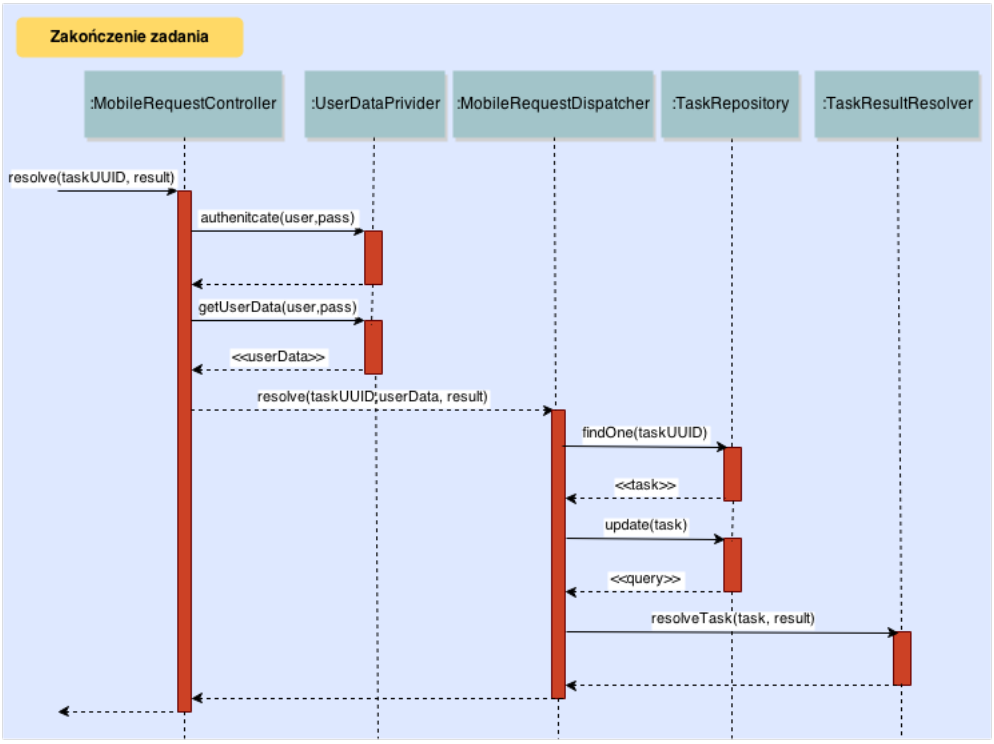
Rysunek 3.9 przedstawia diagram przypisania konkretnego zadania do użytkownika. Przypisanie odbywa się za pomocą REST API. Odpowiednie żądanie HTTP zostaje przyjęte przez *MobileRequestController*, ponownie jak w przypadku pobierania listy zadań, dwie pierwsze operacje wykonywane przez kontroler to autentykacja i pobranie informacji o użytkowniku. Sterowanie zostaje następnie przekazane



Rysunek 3.9: Diagram sekwencji przypisania użytkownika do zadania.

do *MobileRquestDispatcher*'a, który znajduje odpowiednie zadanie w repozytorium, zmienia jego status uzupełniając informacje o przypisanym użytkowniku by następnie zapisać te zmiany. Operacją kompensacyjną do przypisania jest operacja zwolnienia zadania. Diagram sekwencji w przypadku tej drugiej wygląda niemal identycznie jak diagram przypisania z tym że *MobileRequrestDispatcher* przywraca zadaniu status do realizacji.

Na rysunku 3.10 przedstawiono ostatni rodzaj funkcjonalności jakim jest obsługa rozwiązania zadania. Decyzja o rozwiązaniu również pochodzi od strony aplikacji mobilnej i a funkcjonalność ją obsługująca udostępniona jest za pomocą REST API. Tak jak w poprzednich przypadkach przed rozpoczęciem przetwarzania żądania następuje autentykacja oraz pobranie informacji o użytkowniku. Następnie pobierane zostaje zadanie z repozytorium, ustawiany rezultat oraz zmieniany status. Rezultat następnie zostaje odesłany do procesu biznesowego przez klasę *TaskResultResolver*. Jeśli odesłanie rezultatu zakończy się sukcesem zadanie z odpowiednio ustawionym statusem i rezultatem zostaje zapisane do bazy danych.



Rysunek 3.10: Diagram sekwencji rozwiązania zadania.

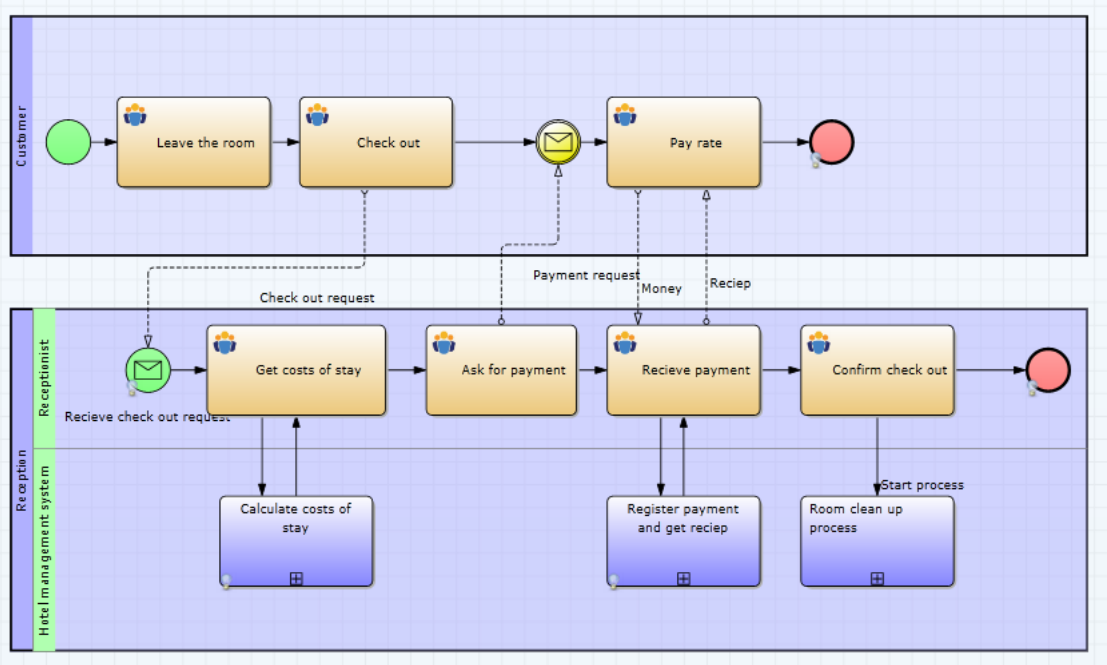
4. Warstwa pośrednia w zastosowaniu

Najlepszym sposobem weryfikacji każdego rozwiązania jest jego konfrontacja z rzeczywistością, przez znalezienie realnych przykładów jego użycia. Nie inaczej jest w przypadku proponowanego w niniejszej pracy rozwiązania.

Pomysłem na przykład jest w tym przypadku proces sprzątania pokoju hotelowego. Celem przykładu jest zademonstrowanie działania warstwy pośredniej do komunikacji procesu biznesowego z aplikacjami mobilnymi, ale również pokazania sposobu integracji takiego rozwiązania z gotowymi systemami zewnętrznymi.

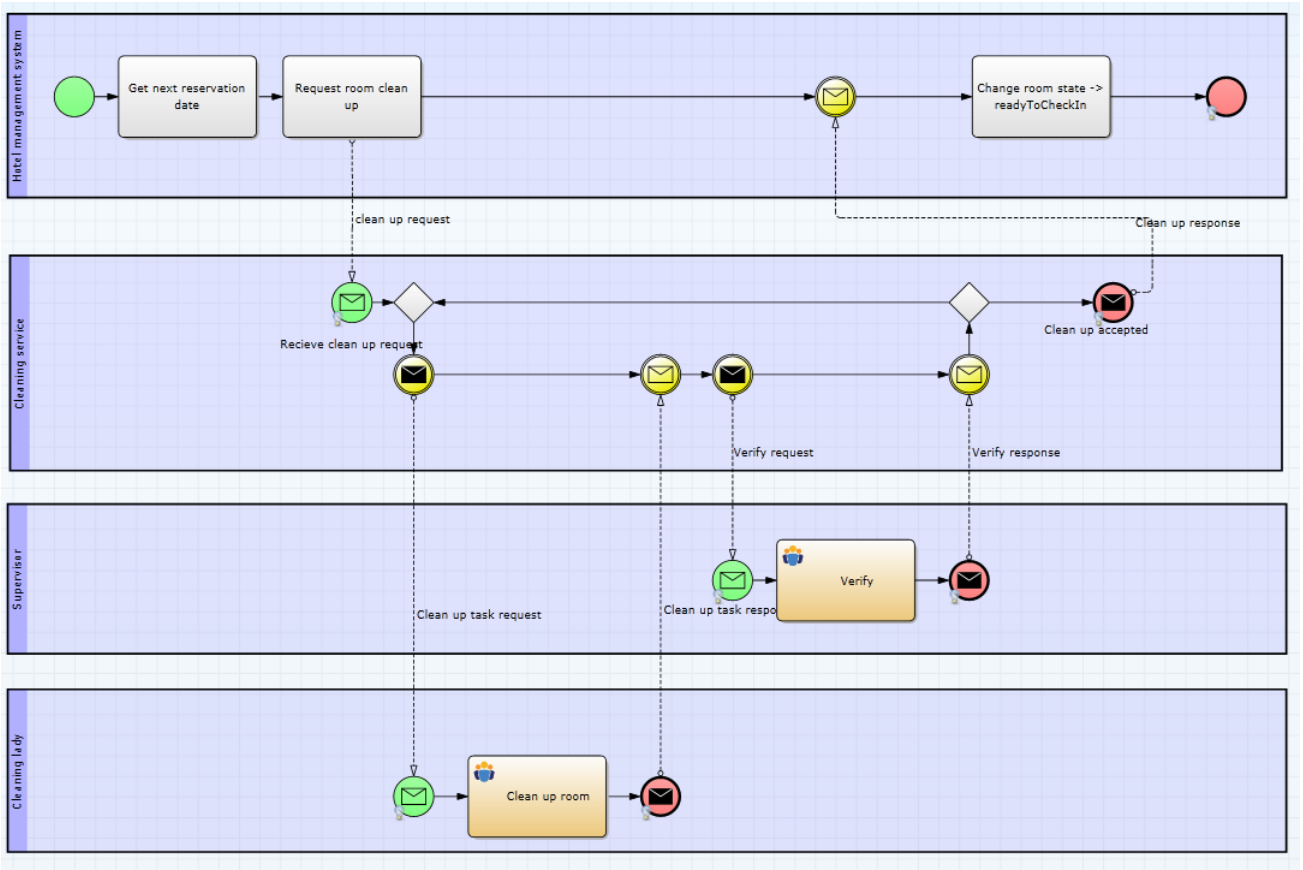
4.1. Przedstawienie koncepcji

Prezentacja przykładu zostanie rozpoczęta od przedstawienia koncepcji rozwiązania w postaci diagramów zaprojektowany za pomocą BPMN 2.0. Przedstawione zostaną dwa diagramy z których pierwszy jest sytuacją w której rozpoczyna się właściwy proces sprzątania hotelu.



Rysunek 4.1: Diagram BPMN procesu wymeldowania klienta z pokoju hotelowego.

Na rysunku 4.1, przedstawiono proces wymeldowania gościa hotelowego z pokoju po zakończonym pobycie. Klient hotelu po opuszczeniu pokoju udaje się do recepcji w celu oddania kluczy i uregulowania płatności. W recepcji przebywa recepcjonista, który odpowiedzialny jest za kontakt z klientem, po otrzymaniu kluczy prosi klienta o zapłatę kosztów pobytu. Po dokonaniu płatności klient hotelowy opuszcza hotel, a recepcjonista oznacza pokój hotelowy jako gotowy do posprzątania, rozpoczynając w ten sposób właściwy proces.



Rysunek 4.2: Diagram BPMN procesu sprzątania pokoju hotelowego.

Rysunek 4.2 przedstawia właściwy proces sprzątania pokoju hotelowego, widać na nim cztery rodzaje aktorów, każdy z nich oznaczony własną linią. Pierwszym z aktorów jest system zarządzania hotelem, jest to miejsce w którym recepcjonista oznacza pokój do posprzątania. System zarządzania hotelem jest miejscem w którym rozpoczyna się proces. Odpowiedzialny jest on za zebranie danych koniecznych do przeprowadzenia sprzątania a następnie przekazanie tych danych do kolejnego aktora, którym jest Serwis Sprzątający. A przypadku realizacji przykładu za pomocą opisanych w niniejszej pracy magisterskiej technologii, aktora tego można utożsamić z procesem BPEL wraz z warstwą pośrednią. Ostani aktor - serwis sprzątający odpowiedzialny jest za przeprowadzenie procesu sprzątania. Proces ten odbywa się przez wysłanie żądania do sprzątaczki (aplikacja mobilna), by po potwierdzeniu posprzątania pokoju wysłać żądanie do nadzorcy (również aplikacja mobilna) który będzie odpowiedzialny za weryfikacje sprzątania. W przypadku braku akceptacji sprzątania przez nadzorcę pokój będzie musiał być kolejny

raz posprzątany. W momencie zatwierdzenia sprzątania, rezultat trafi ponownie do Systemu Zarządzania Hotelem, aby recepcjonista mógł zameldować kolejnych gości.

Na podstawie opisanego powyżej przykładu wyłonić można następujące systemy, których implementacja zostanie dokładniej opisana w dalszej części pracy.

– System Zarządzania Hotelem – Dla celów przykładu, System Zarządzania Hotelem będzie prostą aplikacją internetową odpowiedzialną za udostępnienie formularza dla recepcjonisty oraz wyświetlenie listy rezultatów wykonania procesu.

– Proces BPEL – Będzie to proces zawierający kompletną logikę biznesową odpowiedzialną za przeprowadzenie i weryfikacje sprzątania pokoju hotelowego. Proces ten będzie komunikował się zarówno z warstwą pośrednią jak i z Systemem Zarządzania Hotelem.

– Warstwa pośrednia – Aplikacja internetowa odpowiedzialna za obsługę komunikacji procesu biznesowego z aplikacjami mobilnymi.

– Aplikacja mobilna – aplikacja przeznaczona zarówno dla sprzątaczek jak i dla nadzorców. Udostępniać będzie bardzo prosty interfejs użytkownika służący do wyświetlenia listy zadań oraz do ich przypisywania i rozwiązywania.

4.2. System Zarządzania Hotelem

Zadaniem tej aplikacji jest udostępnienie części funkcjonalność Systemu Zarządzania Hotelem przeznaczonej do oznaczania pokoju hotelowego jako wymagającego posprzątania. Aplikacja udostępniać będzie formularz w którym recepcjonista będzie mógł zgłosić pokój do posprzątania. Formularz będzie gromadził taki dane jak numer pokoju, numer piętra oraz kategorię do której należy pokój. Po zatwierdzeniu formularza dane z niego pochodzące trafią do procesu biznesowego oraz do bazy danych w celu monitorowania postępu procesu. Kiedy proces zakończy swoje działania aplikacja odbierze za pomocą odpowiedniej usługi sieciowej jego rezultat. Odebranie rezultatu skutkować będzie zmianą statusu pokoju. Aplikacja udostępniać będzie również listę pokoi wraz z ich statusami w postaci strony www.

4.2.1. Wybrane technologie

Aplikacja zostanie zrealizowana przy wykorzystaniu technologii:

– język programowania Java

– Spring MVC

– Bootstrap Framework – jest to zbiór klas css oraz biblioteka stworzona w języku JavaScript, służący do tworzenia interfejsu użytkownika przy pomocy zaawansowanych kontrolek html.

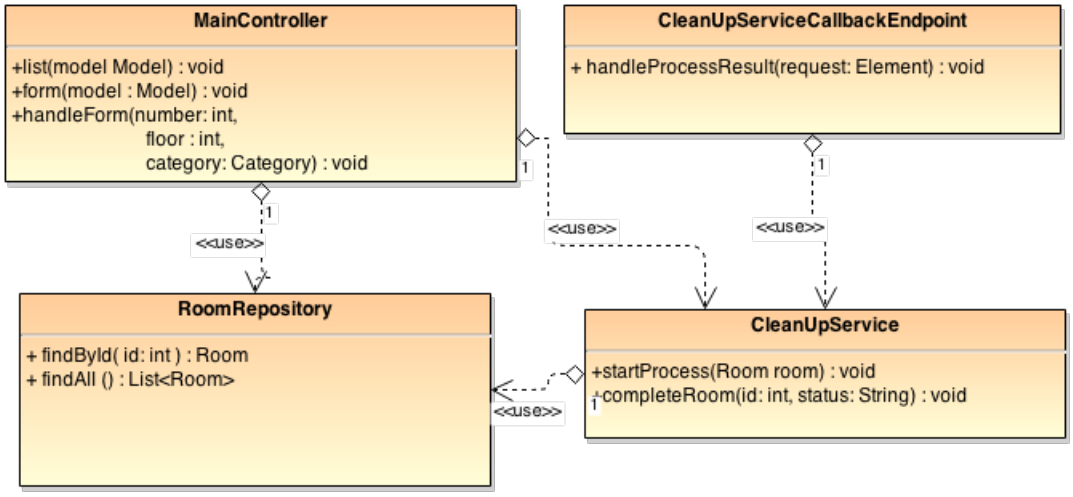
– Maven

– Spring Web Services

– Spring Data oraz Hibernate - jeden z najbardziej popularnych narzędzi ORM, przeznaczonych na platformę Java.

– H2 in memory - baza danych której cykl życia jest równoznaczny z cyklem życia aplikacji. Jest ona tworzona podczas uruchamiana aplikacji, dane znajdujące w się w niej są przechowywane w pamięci podręcznej. Bazy tego typu wykorzystywane są przede wszystkim do implementacji testów jednostkowych. Zdecydowanym plusem skorzystania z tego rodzaju bazy danych jest brak konieczności instalacji dodatkowych narzędzi co w przypadku niniejszego przykładu jest dużym udogodnieniem.

4.2.2. Implementacja



Rysunek 4.3: Diagram klas systemu zarządzania hotelem.

Na rysunku 4.3 został przedstawiony diagram klas Systemu Zarządzania Hotelem. Znajdują się na nim jedynie najbardziej istotne klasy z punku widzenia tej implementacji. Klasa *MainController* jest odpowiedzialna za udostępnienie interfejsu użytkownika oraz za obsługę żądań z niego pochodzących. Udostępnia ona trzy metody:

– *list* – metoda za pomocą RomRepository pobiera z bazy danych wszystkie pokoje, które zostały zgłoszone do posprzątania. Pobrana lista jest następnie umieszczana w dynamicznej stronie www za pomocą technologii JSP.

– *form* – metoda odpowiedzialna za wyświetlenie formularza w postaci html dla użytkownika.

– *hadleForm* – metoda ta jest uruchamiana w przypadku zatwierdzenia danych wprowadzonych do formularza przez użytkownika. Metoda dodaje nowy pokój do bazy danych oraz uruchamia proces biznesowy z wykorzystaniem klasy *CleanUpService*.

Druga bardzo ważną klasą jest *CleanUpServiceCallbackEndpoint*, jest to klasa stworzona z wykorzystaniem technologii Spring Web Services. Klasa ta obsługuje zdefiniowaną w konfiguracji usługę sieciową. Usługa służy do odbierania rezultatów wykonania procesu biznesowego. Jej rolą jest przeczytania komunikatu przysłanego przez proces biznesowy by następnie zmienić status sprzątanego pokoju za pomocą klasy *CleanUpService*.

4.2.3. Uruchomienie

Aplikacja jest kompilowana do postaci pliku o rozszerzeniu war. Pliki tego typu mogą być uruchamiana na dowolnym serwerze aplikacyjnym Java, np. Tomcat. Aplikacja powinna zostać uruchomiona na porcie 8181, ponieważ pozostałe aplikacje będą od niej tego wymagały. Np. proces biznesowy swój rezultat będzie wysyłał właśnie do lokalnej maszyny na ten port.

4.3. Proces BPEL

Proces BPEL jest odpowiedzialny za obsługę logiki biznesowej związanej z przeprowadzeniem procesu sprzątania. Proces służy w pewnym sensie jako narzędzie integracyjne aplikację internetową jaką jest System Zarządzania Hotelem z aplikacjami mobilnymi.

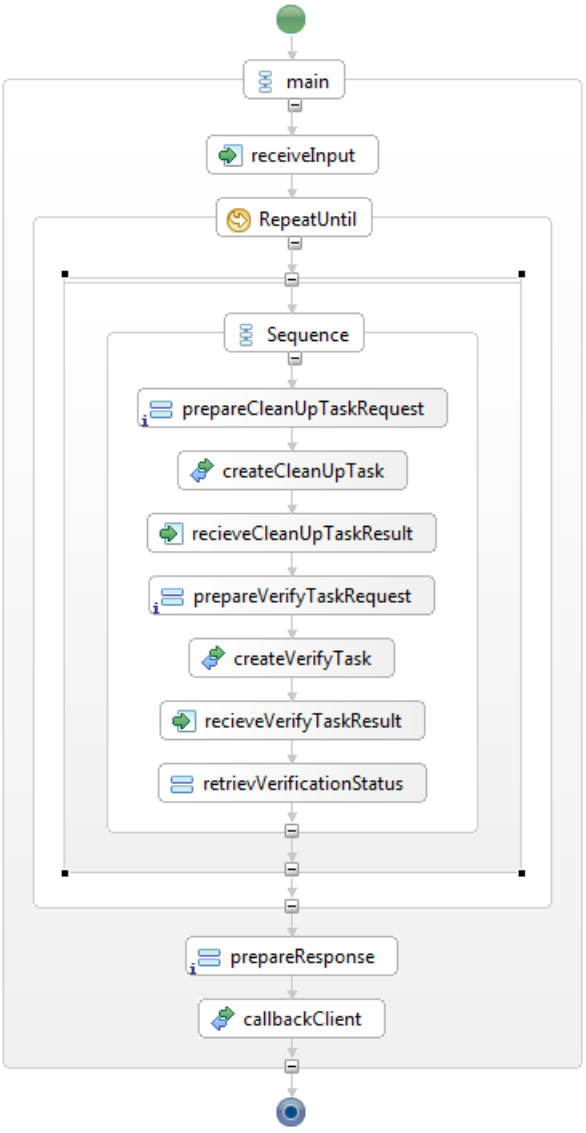
Przebieg działania procesu rozpoczyna się w Systemie Zarządzania Hotelem, który uruchamia usługę sieciową udostępnianą jako punkt dostępowy procesu. Proces po otrzymaniu żądania z systemu zewnętrznego tworzy nowe zadanie przeznaczone dla osoby sprzątającej pokój by następnie przejść w tryb oczekiwania na jego rezultat. Gdy pokój zostanie posprzątany, proces tworzy kolejne zadanie tym razem przeznaczone dla nadzorcy i ponownie przechodzi w tryb oczekiwania na jego rezultat. Gdy nadzorca zakończy swoje zadanie proces sprawdzi jego rezultat, jeśli sprzątanie pokoju zostało zaakceptowane proces zakończy swoje działanie poprzez wysłanie rezultatu do Systemu Zarządzania Hotelem. Gdy sprzątanie pokoju nie zostanie zaakceptowane proces będzie tworzył kolejne zadania dla osoby sprzątającej do momentu zatwierdzenia przez nadzorcę.

4.3.1. Implementacja

Do implementacji procesu BPEL została wykorzystana wtyczka do zintegrowanego środowiska programistycznego Eclipse. Wtyczka ta udostępnia narzędzia do modelowania procesów BPEL w sposób graficzny. Na rysunku 4.4 został przedstawiony zrzut ekranu przedstawiający implementację niniejszego procesu biznesowego. Na rysunku widać doskonale kolejność wywoływania poszczególnych aktywności procesu BPEL, nie widać natomiast zastosowania mechanizmu korelacji oraz zakresu.

4.3.1.1. Korelacja

Mechanizmem na który w szczególności należy zwrócić uwagę podczas opisu procesu biznesowego jest mechanizm korelacji. Mechanizm ten służy do powiązania dwóch lub więcej operacji wewnątrz jednej instancji procesu. W przypadku niniejszego przykładu proces korelacji wykorzystywany jest do



Rysunek 4.4: Wizualizacja procesu BPEL w postaci graficznej za pomocą wtyczki do Eclipsa.

powiązywania operacji stworzenie zadania z operacją odebrania rezultatu zadania. Zazwyczaj środowisko uruchomieniowe posiada więcej niż jedną instancje procesu, w momencie gdy zostaje odebrany rezultat zadania np. sprzątania środowisko powinno w jakiś sposób zidentyfikować instancję procesu do której zaadresowany jest ten rezultat.

Mechanizm korelacji realizowany jest za pomocą tak zwanych zbiorów korelacji. Zbiór korelacji nie jest niczym innym jak zbiorem zmiennych. Zmienne te inicjalizowane są przy pierwszym użyciu dowolną wartością, podczas kolejnego użycia służą do odszukania instancji procesu o zadanej wartości.

W niniejszym procesie istnieją dwa zbiory korelacji, zdefiniowane wewnątrz ciała pętli:

```
<bpel:correlationSets>
  <bpel:correlationSet name="cleanUpTask"
      properties="tns:taskUUID"/>
```

```
  <bpel:correlationSet name="verifyTask"
      properties="tns:verifyTaskUUID"/>
</bpel:correlationSets>
```

Listing 4.1: Definicja zbiorów korelacji.

Zbiory te wykorzystywane są do powiązania operacji stworzenia i odbioru rezultatu zadania sprzątanie i weryfikacja.

```
<bpel:receive name="recieveCleanUpTaskResult" partnerLink="client"
  operation="cleanUpTaskCallback" portType="tns:cleanUpProcess"
  variable="clientRequest">
            <bpel:correlations>
                <bpel:correlation set="cleanUpTask" initiate="no">
       </bpel:correlation>
            </bpel:correlations>

</bpel:receive>
```

Listing 4.2: Wykorzystanie zbiorów korelacji w aktywności invoke.

Operacja invoke odpowiedzialna za stworzenie zadania inicjalizuje zbiór korelacji odebraną z warstwy pośredniej wartością unikalnego identyfikatora zadania.

```
<bpel:receive name="recieveCleanUpTaskResult" partnerLink="client"
    operation="cleanUpTaskCallback"
            portType="tns:cleanUpProcess" variable="clientRequest">
            <bpel:correlations>
                  <bpel:correlation set="cleanUpTask" initiate="no">
    </bpel:correlation>
            </bpel:correlations>
</bpel:receive>
```

Listing 4.3: Wykorzystanie zbiorów korelacji w aktywności receive.

Operacja recieve następnie na podstawie tego samego unikalnego identyfikatora odnajduje odpowiednią instancje by kontynuować jej działanie.

4.3.1.2. Zakres

Bezpośrednio powiązany z mechanizmem korelacji jest mechanizm zakresów (ang. scope). Służy on do odizolowania dwóch lub więcej operacji wewnątrz procesu. Izolacja ta może zostać wykorzystana np. do odpowiedniego zarządzania sytuacjami wyjątkowymi. W przypadku niniejszego procesu mechanizm zakresu został zastosowany do odizolowania kolejnych wywołań pętli w celu uzyskania efektu ponownej inicjalizacji zbiorów korelacji w każdym przebiegu pętli. W przypadku nie zastosowania mechanizmy zakresu, podczas pierwszego wywołania przebiegu pętli zbiór korelacji zostałby zainicjalizowany identyfikatorem pierwszego zadania. W przypadku kolejnych wywołań pętli metoda recieve oczekiwała by na identyfikator pierwszego zadania mimo, że zostało ono już rozwiązane a kolejny przebieg pętli stworzył następne zadanie.

Mechanizm zakresu stosuje się obejmując izolowane elementy w drzewie xml elementem *scope*.

4.3.2. Uruchomienie

Proces BPEL został stworzony zgodnie ze specyfikacją BPEL, może być zatem wdrożony na dowolne środowisko uruchomieniowe dla procesów biznesowych. W przykładzie zastosowano środowisko Apache ODE.

Apache ODE jest również aplikacją internetową napisaną w języku Java i może być uruchomiony z wykorzystaniem dowolnego serwera aplikacyjnego. Ważne jest aby silnik Apache ODE został uruchomiony z wykorzystaniem portu 8080. System Zarządzania Hotelem będzie próbował uruchomić proces na lokalnej maszynie pod tym właśnie portem.

4.4. Warstwa pośrednia

Najbardziej istotną częścią przykładu z punktu widzenia niniejszej pracy magisterskiej jest warstwa pośrednia. Jest ona odpowiedzialna za odbiór żądań od procesu biznesowego i przekazanie ich do aplikacji mobilnych. Warstwa pośrednia udostępnia w tym celu dwie usługi sieciowe. Jedną do utworzenia zadania - sprzątanie pokoju i drugą do utworzenia zadania - weryfikacja sprzątania. Żądania odbierane przez usługi sieciowe zapisywane są następnie w dokumentowej bazie danych i oczekują na realizacje przez użytkowników mobilnych. Aplikacje mobilne komunikują się warstwą pośrednią za pomocą REST API. Są w stanie w ten sposób zautoryzować się, pobrać listę zadań i rozwiązywać je. Zadania po rozwiązaniu odsyłane są do procesów biznesowego z wykorzystaniem odpowiednich usług sieciowych.

4.4.1. Wybrane technologie

Warstwa pośrednia z racji tego, że wykorzystuje przedstawioną w niniejszej pracy magisterskiej bibliotekę, została zrealizowana w technologii Spring i korzysta z bazy danych Mongo DB.

4.4.2. Implementacja

Jak wspomniano wcześniej warstwa pośrednia wykorzystuje opisaną w niniejszej pracy magisterskiej bibliotekę stworzoną do tego celu. Opis implementacji zaczniemy od dwóch najważniejszych plików konfiguracyjnych tej biblioteki. Pierwszym z nich jest plik bpel4mobile.properties znajdujący się w katalogu resources. W pliku tym znajdują się dane do komunikacji z bazą danych MondoDB, oraz ścieżka do drugiego pliku konfiguracyjnego - humanInteractions.xml. Plik xml jest najważniejszym plikiem konfiguracyjnym z punktu widzenia stworzonej biblioteki, zawiera od definicje zadań, grup użytkowników i przypisania poszczególnych grup do zadań.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <htd:humanInteractions
3   xmlns:htd="http://docs.oasis-open.org/bpel4people/ws-humantask"
```

```
4   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
6   xsi:schemaLocation="http://docs.oasis-open.org/bpel4people/ws-humantask http://
      docs.oasis-open.org/bpel4people/ws-humantask.xsd">
7
8   <htd:logicalPeopleGroups>
9     <htd:logicalPeopleGroup name="cleaningLadies">
10      <htd:parameter name="correspondingFloor" type="xsd:int"/>
11    </htd:logicalPeopleGroup>
12    <htd:logicalPeopleGroup name="supervisors" />
13  </htd:logicalPeopleGroups>
14  <htd:tasks>
15    <htd:task name="cleanUpTask">
16      <htd:priority>5</htd:priority>
17      <htd:peopleAssignments>
18        <htd:potentialOwners>
19          <htd:from logicalPeopleGroup="cleaningLadies">
20          <htd:argument name="correspondingFloor">
21            eq:request/room/floor
22          </htd:argument>
23          </htd:from>
24        </htd:potentialOwners>
25      </htd:peopleAssignments>
26    </htd:task>
27    <htd:task name="verifyTask">
28      <htd:priority>5</htd:priority>
29      <htd:peopleAssignments>
30        <htd:potentialOwners>
31        <htd:from logicalPeopleGroup="supervisors" />
32        </htd:potentialOwners>
33      </htd:peopleAssignments>
34    </htd:task>
35  </htd:tasks>
36 </htd:humanInteractions>
```

Listing 4.4: Plik konfiguracyjny z opisem zadań warstwy pośredniej.

Na powyższym fragmencie kodu widoczny jest pliku konfiguracyjny niniejszego przykładu. Linijki 8-13 zawierają definicję dwóch grup użytkowników, jednej dla sprzątaczek i drugiej dla nadzorców. Sprzątaczki posiadają dodatkowy atrybut którym jest numer piętra na którym pracują. W elemencie *tasks* znajdują się definicję zadań, widać na nich, że zadanie sprzątania przypisane jest do sprzątaczek pracujących na danym piętrze, natomiast weryfikację mogą przeprowadzać dowolni nadzorcy. Sposób konfiguracji zadań jest bardzo prosty, przejrzysty i elastyczny.

Kolejną ważną częścią implementacji warstwy pośredniej jest udostępnienie usług sieciowych dla procesu biznesowego. Zadanie to zrealizowane zostaje przy pomocy modułu Spring WS. Spring WS do tworzenia usług sieciowych preferuje podejście interfejs najpierw (ang. contract first), w celu udostępnie-

nia usług sieciowej konieczne jest zdefiniowanie pliku ze schematem komunikatów xml. Powstają w tej sposób dwa pliki dla dwóch różnych usług sieciowych. Pliki te wykorzystane zostaną do wygenerowanie odpowiednich kontraktów WSDL przez Spring WS. Po stronie kodu Java, powstają odpowiedniki opisanych komunikatów w postaci klas. Po dwie klasy na usługę sieciową do obsługi wiadomości wejściowej i rezultatu. Przykładowo dla weryfikacji sprzątania klasy będą wyglądać następująco:

```
@XmlRootElement(name="request", namespace= XMLNamespace.VERIFY)
public class VerifyRequest {

  @XmlElement(name="deadline",
    namespace=XMLNamespace.VERIFY)
  private Date deadline;

  @XmlElement(name="cleanUpPerformer",
    namespace=XMLNamespace.VERIFY)
  private String cleanUpPerformer;

  @XmlElement(name="room",
    namespace=XMLNamespace.VERIFY)
  private Room room;
}

@XmlRootElement(name="result", namespace="http://bpel4mobile.com/example/hotel/
    schemas")
public class VerifyResponse {

  public enum Status {
    success, toRepeat
  }

  @XmlElement(name="status", namespace="http://bpel4mobile.com/example/hotel/
    schemas")
  private Status status;
}
```

Listing 4.5: Klasy zawierające komunikaty wejściowe i wyjściowe usługi weryfikacji sprzątania.

Spring WS oprócz przygotowania plików ze schematem xml komunikatów, wymaga zdefiniowanie servletu w plikach xml. Servlet ten wskazuje na pliki ze schematem xml oraz definiuje przestrzeń nazw i port usługi sieciowej.

```
<beans ... >
  <sws:dynamic-wsdl id="verifyService"     portTypeName="verifyServicePort"
    locationUri="http://localhost:8282/hotel-cleanup-mobile-middleware/ws/
    verifyService"
    targetNamespace="http://bpel4mobile.com/ schemas/example/verifyService">
  <sws:xsd location="/WEB-INF/schema/verify-request.xsd"/>
  </sws:dynamic-wsdl>
```

```
</beans>
```

Listing 4.6: Servlet usługi weryfikacji sprzątania.

Ostatnią rzeczą do implementacji w celu udostępnienia usługi sieciowej jest przygotowanie odpowiedniej klasy Java która będzie punktem końcowym zdefiniowanego wcześniej servletu. Klasa ta będzie uruchamiać odpowiednią instancję serwisu do obsługi zadań zdefiniowanych w warstwie pośredniej. Dalszą obsługą zadania wraz z odesłaniem odpowiedzi zajmie się przygotowana w poprzednim rozdziale biblioteka.

```
@Endpoint
public class VerifyServiceEndpoint {

  @Autowired
  @Qualifier(value="verifyTaskService")
  private TaskService tasService;

  @PayloadRoot(namespace = XMLNamespace.VERIFY, localPart = "VerifyTaskRequest")
  @ResponsePayload
  public Element handleHolidayRequest(@RequestPayload Element request) throws
    Exception {
    return tasService.handleTaskRequest(request, VerifyRequest.class,
    VerifyResponse.class);
  }
}
```

Listing 4.7: Punkt końcowy usługi weryfikacji sprzątania.

Do zakończenia implementacji warstwy pośredniej konieczne jest jeszcze dostarczenie informacji o użytkownikach systemu. Biblioteka utworzona w poprzednim rozdziale przewiduje do tego celu odpowiedni interfejs, który zostanie zaimplementowany. Na potrzeby niniejszego przykładu nie konieczne jest tworzenie adaptera do pobierania danych o użytkownikach z bazy danych czy innych systemów katalogowych, utworzymy prostą klasę dostarczającą statyczne dane.

```
@Service
public class UserDataProviderImpl extends AbstractUserDataProvider {

  private Map<String, UserData> users = new HashMap<String, UserData>();

  @PostConstructor
  public void init(){
    UserGroupData cleanUpGroupDataFloor1 = new UserGroupData();
    cleanUpGroupDataFloor1.setName("cleaningLadies");
    cleanUpGroupDataFloor1.getArguments().put("correspondingFloor", 1);

    UserGroupData cleanUpGroupDataFloor2 = new UserGroupData();
    cleanUpGroupDataFloor2.setName("cleaningLadies");
    cleanUpGroupDataFloor2.getArguments() .put("correspondingFloor", 2);
```

```
15
16      UserGroupData supervisorGroup = new UserGroupData();
17      supervisorGroup.setName("supervisors");
18
19      UserData jadzia = new UserData();
20      jadzia.setUsername("Jadzia");
21      jadzia.getGroups().add(cleanUpGroupDataFloor1);
22      users.put("Jadzia", jadzia);
23
24      UserData stasia = new UserData();
25      stasia.setUsername("Stasia");
26      stasia.getGroups().add(cleanUpGroupDataFloor2);
27      users.put("Stasia", stasia);
28
29      UserData zdzislaw = new UserData();
30      zdzislaw.setUsername("Zdzislaw");
31      zdzislaw.getGroups().add(supervisorGroup);
32      users.put("Zdzislaw", zdzislaw);
33    }
34    @Override
35    public boolean authenticate(String username,
36          String password) {
37      return (users.contains(username) && "password".equals(password));
38    }
39
40    @Override
41    public UserData getUserData(String username, String password) {
42      return users.get(username);
43    }
44  }
```

Listing 4.8: Dostawca informacji o użytkownikach.

4.4.3. Uruchomienie

Aplikacja budowana jest za pomocą narzędzia maven. Plikiem wynikowym podobnie jak w przypadku systemu zarządzania hotelem jest plik z rozszerzeniem .war, który można uruchomić na dowolnym serwerze aplikacyjnym java. Warstwa pośrednia aby współgrała z pozostałymi systemami, musi zostać uruchomiona na porcie 8282.

4.5. Aplikacja mobilna

Aplikacja mobilna jest częścią przykładu odpowiedzialną za rozwiązywanie zadań poprzez udostępnienie użytkownikom odpowiedniego interfejsu. Aplikacja komunikuje się w warstwą pośrednią za po-

mocą REST API w celu pobrania listy zadań, operacji na tych zadaniach i wysłania rezultatu. Aplikacja została zaimplementowana na najbardziej popularną platformę mobilną - Android.

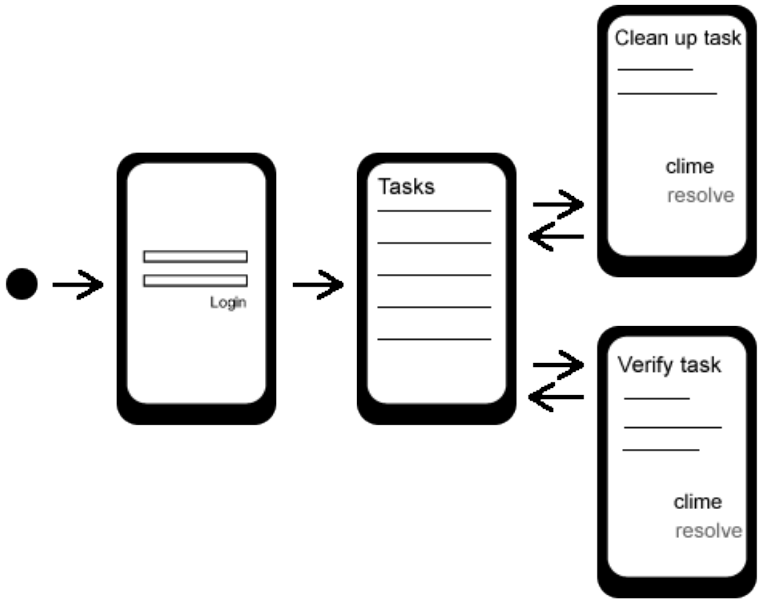
Aplikacja wykorzystuje specyficzny dla tej platformy mechanizm synchronizacji zarządzany przez system operacyjny. Synchronizacja uruchamiana jest bez konieczności działania aplikacji w momencie gdy urządzenie ma dostęp do sieci. Przez synchronizacje w tym wypadku rozumiane jest pobranie listy zadań dostępnych dla użytkownika i ich zapis w lokalnej bazie danych (SQLite). Aplikacja przewiduje możliwość logowania więcej jak jednego użytkownika, dane każdego z nich zapisywane są do osobnej bazy danych. Użytkownik po wejściu do aplikacji widzi listę dostępnych dla niego zadań mimo, że może nie mieć dostępu do sieci. Użytkownik może zadeklarować chęć wykonania zadania oraz w momencie gdy jest do niego przypisany rozwiązać je klikając odpowiednie przyciski.

4.5.1. Wybrane technologie

Aplikacja została zrealizowana z wykorzystaniem język Java, platformy Android, bibliotek Spring for Android oraz OrmLite.

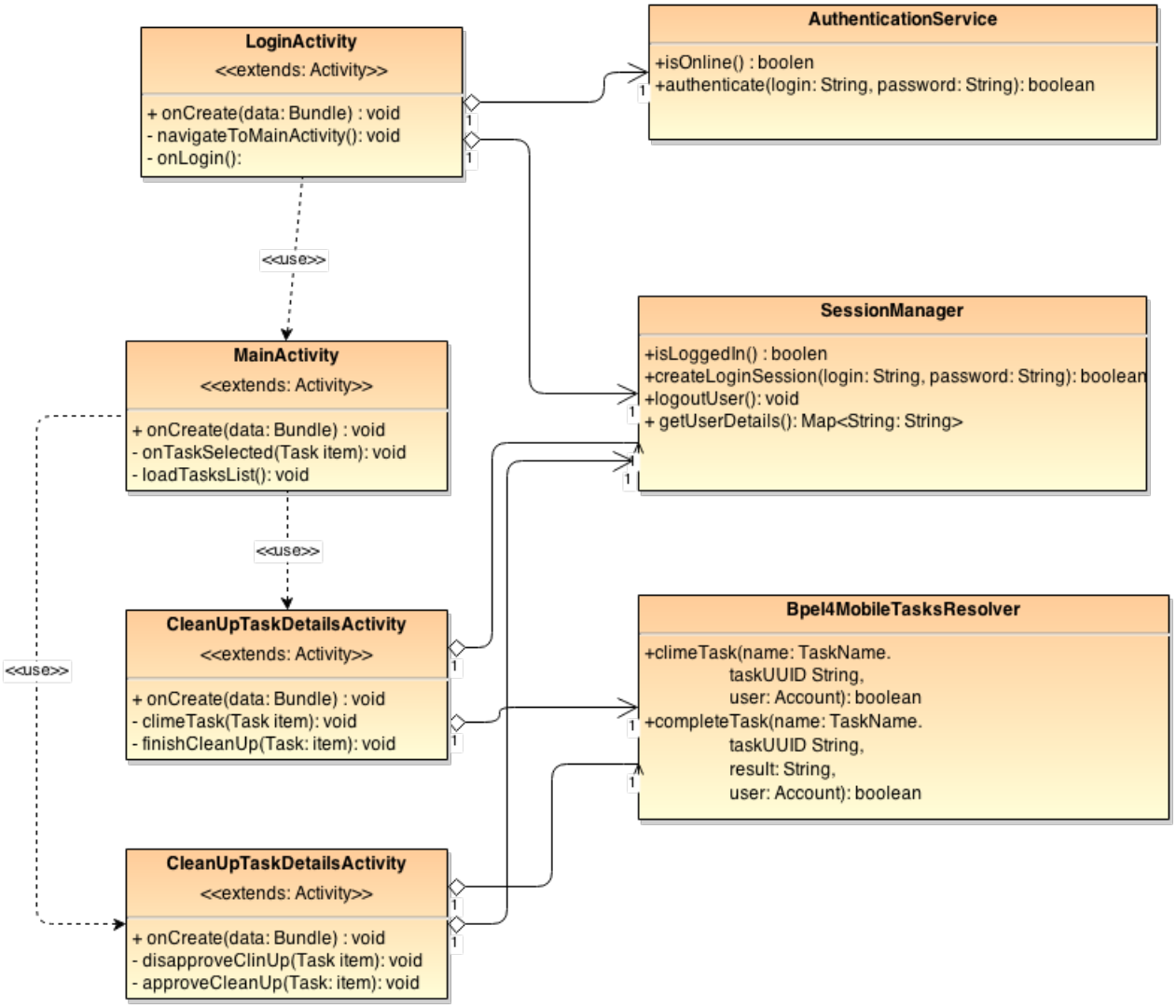
4.5.2. Implementacja

Implementacja aplikacji mobilnej bierze pod uwagę dwie kwestie. Pierwszą z nich jest udostępnienie interfejsu użytkownika do logowania, wyświetlenia listy zadań i ich szczegółów. Kwestia druga to synchronizacja listy zadań z wykorzystaniem interfejsu przygotowanego do tego celu przez platformę android. Wykorzystanie API do synchronizacji gwarantuje możliwość pracy z zadaniami w trybie offline.



Rysunek 4.5: Ekrany w aplikacji mobilnej wraz z przepływem komunikacji.

Na rysunku 4.5 przedstawiono przepływ komunikacji pomiędzy ekranami aplikacji. Po starcie domyślnie uruchamiany jest ekran logowania, który po poprawnym zalogowaniu otwiera główny ekran aplikacji czyli listę zadań. Z listy zadań w zależności od wybranego typu zadania użytkownik przenoszony jest do szczegółów, gdzie może wykonać akcję przypisania do zadania lub jego rozwiązania jeśli jest przypisany.

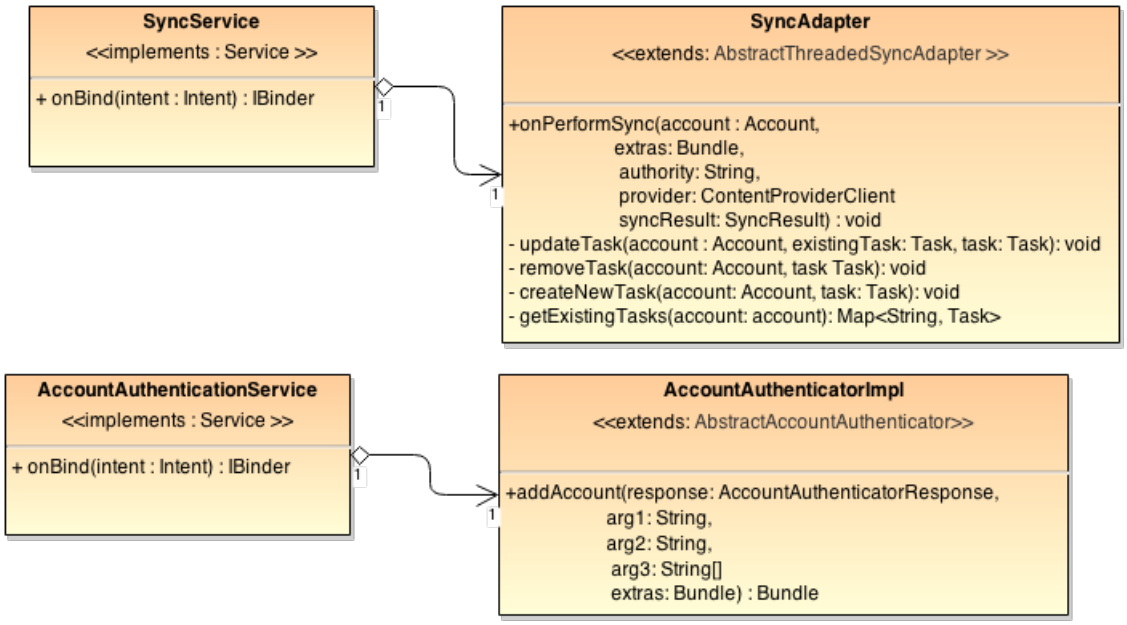


Rysunek 4.6: Diagram klas odpowiedzialnych za działanie poszczególnych ekranów.

Więcej szczegółów implementacyjnych poszczególnych ekranów widać na rysunku 4.6, przedstawia on diagram klas poszczególnych ekranów oraz serwisów z których te klasy korzystają. Ekran logowania korzysta z serwisu do autoryzacji, który autoryzuje użytkownika wysyłając żądanie do warstwy pośredniej. Drugim z serwisów wykorzystywanych przez ekran logowania jest zarządca sesji, który odpowiada za utworzenie sesji użytkownika, w celu przechowywania jego danych dla pozostałych ekranów. Ekran główny wyszukuje listę zadań użytkownika bezpośredni z bazy danych, nie korzysta z innych serwisów.

Ekrany szczegółów poszczególnych zadań wykorzystuje zarówno zarządcę sesji jak i serwis przeznaczony do komunikacji z warstwą pośrednią w celu przypisania lub rozwiązania zadania.

W celu rozwiązania problemu synchronizacji listy zdań, skorzystano z interfejsów udostępnianych przez platformę android. Każda aplikacji przeznaczona na tą platformę posiada główny plik konfiguracyjny zwany *AndroidManifest.xml*. W pliku tym zostały wskazane dwa serwisy implementujące interfejs serwisu udostępniony przez platformę, odpowiedzialne za udostępnienie adaptera do synchronizacji i implementacje zarządcy kont użytkowników.



Rysunek 4.7: Diagram klas odpowiedzialnych za synchronizację.

Rysunek 4.7 przedstawia diagram wyżej wymienionych klas. Cykl życia adaptera synchronizacji, udostępnionego przez odpowiedni serwis, zarządzany jest przez system operacyjny. Proces synchronizacji uruchamiany jest cyklicznie na podstawie danych statystycznych o ilości zsynchronizowanych rekordów zwracanych przez adapter. Synchronizacja uruchamiana jest tylko w momencie dostępu aplikacji do Internetu, bez konieczności jej uruchamiania. Dzięki takiemu działaniu adaptera lista zadań aktualizowana jest mimo że użytkownik nie korzysta z aplikacji i jest zawsze aktualna kiedy użytkownika zechce z niej korzystać mimo braku dostępy do Internetu.

Implementacja zarządcy użytkownikami, jest konieczna do utworzenia kona użytkownika w systemie operacyjnym. Konto to jest wykorzystywane do synchronizacji.

4.5.3. Uruchomienie

Aplikacja podobnie jak pozostałe systemy jest budowana przy pomocy narzędzia maven. Po poprawnym zbudowaniu aplikacji powstaje plik o rozszerzeniu apk, który może być bezpośrednio zainstalowany na urządzeniu mobilnym z systemem android.

5. Podsumowanie

Celem niniejszej pracy magisterskiej była analiza możliwości adaptacji środowisk uruchomieniowych na aplikacjach mobilnych. Cel ten został zrealizowany poprzez przedstawienie podstawowej analizy dwóch wyżej wymienionych technologii, oraz próbę ich zestawienia. W niniejszej pracy magisterskiej zostały zaproponowane dwa różne podejścia do problemu. Pierwsze z nich - podejście adaptacyjne okazało się błędne ze względu na charakterystykę aplikacji mobilnych. Drugie podejście - integracyjne okazuje się znakomitym pomysłem w połączeniu z technologią BPEL. Głębsza analiza rozwiązania skutkuje powstaniem idei wykorzystania warstwy pośredniej odpowiedzialnej za adaptacje specyficznej charakterystyki aplikacji mobilnych do współpracy z procesami biznesowymi w środowisku usług sieciowych. Idea przełożona została na projekt, który został opisany w dalszych rozdziałach niniejszej pracy magisterskiej i implementację. Implementacja ta posłużyła do przygotowania przykładowego systemu, który kompleksowo pokrywa wszystkie części niezbędne do zastosowania warstwy pośredniej w komercyjnych systemach. Przedstawiony system oprócz przykładu zastosowania traktować można jako wytyczne projektowaniu tego rodzaju aplikacji. Dostrzec w nim można jak w praktyce funkcjonuje podejście top-down w projektowaniu systemów rozproszonych za pomocą procesów biznesowych.

Wynikiem pracy magisterskiej okazała się być biblioteka klas umożliwiająca zastosowanie przedstawionej koncepcji integracji procesów biznesowych z aplikacjami mobilnymi w komercyjnych zastosowaniach. Biblioteka ta w celu lepszej dystrybucji zadań, która powinna przełożyć się na skuteczniejszej rozwiązywanie problemów może zostać zintegrowania z systemami typu Contex Aware Middleware. Wykonanie takiej integracji możliwe jest przez zastosowanie odpowiednich interfejsów których implementacja może zostać dostosowana do potrzeb konkretnego rozwiązania.

Biblioteka ponadto może zostać rozszerzona o dodatkowe funkcjonalności jak eskalacja w przypadku zwlekania z rozwiązywaniem zadań oraz notyfikacje o zadaniach do rozwiązania.

Bibliografia

[1] European Association of Business Process Management. *Organization home website*. http://www.eabpm.org/.

[2] Antipodes. *What is a Business Process Management system?*. http://www.antipodes.bg/en/cubes/what_is_bpm/.

[3] 2008 Cisco Systems, Inc. *Cisco Context-Aware Mobility Solution*. Solution Overview

[4] OASIS. *OASIS Web Services Business Process Execution Language (WSBPEL) TC* https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel

[5] OASIS *Web Services Business Process Execution Language Version 2.0*. OASIS Standard 11 April 2007

[6] Mulesoft. *What is application orchestration?* http://www.mulesoft.org/what-application-orchestration

[7] Object Management Group *Business Process Model and Notation* http://www.bpmn.org/

[8] Apache Software Foundation *Apache ODE Home Page* http://ode.apache.org/

[9] IBM and SAP. *WS-BPEL Extension for People – BPEL4People*. A Joint White Paper by IBM and SAP July 2005

[10] Daniel Romero *Context-Aware Middleware: An overview* Paradigma 2, 3 (2008)

[11] Patryk Yarpo Jar *REST – ciekawszy sposób na komunikację client-server* http://www.yarpo.pl/2012/07/29/rest-ciekawszy-sposob-na-komunikacje-client-

[12] Spring community *Spring Framework - home page* http://projects.spring.io/spring-framework/

[13] Spring community *Spring Framework - Web Services component*
`http://projects.spring.io/spring-ws/`

[14] Rod Johnson, Juergen Hoeller, Alef Arendsen, Thomas Risberg, Colin Sampaleanu *Spring Framework. Profesjonalne tworzenie oprogramowania w Javie*
`Helion 2006-06-14`

[15] MongoDB community *Mongo DB documentation*
`http://docs.mongodb.org/manual/`

[16] 2002-2014 The Apache Software Foundation *Apache Maven Project*
`http://maven.apache.org/`