Integracja technologii mobilnych i systemów klasy Enterprise

Wojciech Klicki Konrad Starzyk

Spis treści

Rozdział 1

Integracja w środowiskach Enterprise

Oprogramowanie Enterprise jest pojęciem dość szerokim, opisującym systemy przeznaczone dla przedsiębiorstw, których działanie odwzorowuje zachodzące procesy biznesowe. Niekiedy pojęcie to odnosi się do oprogramowania pisanego na zamówienie lub też do rozbudowanych pakietów wspierających określone czynności, np. kontakty z klientami lub księgowość. Rzadko się jednak zdarza, by istniał jeden system który potrafiłby spełnić wymagania klienta, a nawet jeżeli, z różnych przyczyn firma może nie chcieć go wdrożyć. Tak więc nawet w jednym przedsiębiorstwie często można się spotkać z sytuacją w której działa wiele niezależnych aplikacji, nierzadko pisanych przez różne firmy, które muszą się ze sobą komunikować w celu wymiany danych. Jeszcze do nie dawna mówiąc o integracji mieliśmy na myśli właściwie wyłącznie serwery aplikacyjne. Obecnie obowiązującym trendem jest postępująca miniaturyzacja i wzrost mocy obliczeniowej urządzeń mobilnych, które udostępniają klientom usługi dostępne do tej pory wyłącznie za pośrednictwem komputera stacjonarnego. Tworząc aplikację na urządzenia przenośne, która

będzie współpracowała z istniejącymi już systemami, napotykamy jednak na problem komunikacji.

1.1 Istniejące podejścia do integracji

Rozważając możliwe sposoby integracji technologii mobilnych z rozbudowaną aplikacją klasy Enteprise musimy zwrócić uwagę na udostępniane z obu stron mechanizmy.

Transfer plików W tym przypadku dane są zapisywane i odczytywane z plików. Niezbędna jest wspólna przestrzeń dyskowa lub sposób przesyłania (np. FTP). Z tego powodu musimy odrzucić takie rozwiązanie jako niedające się zaimplementować w środowisku mobilnym.

Wspólna baza danych Dane są składowane we wspólnej bazie danych. Ze względu na zawodność transferu oraz długi czas przesyłania danych pomiędzy urządzeniem mobilnym a serwerem, a tym samym konieczność długiego oczekiwania, rezygnujemy z takiego rozwiązania.

RPC Zdalne wywoływanie procedur (ang. Remote Procedure Call) jest mechanizmem który pozwala wykonywać procedury udostępniane przez zdalny system. Założeniem wywołań RPC jest synchroniczność, która wprawdzie nie oznacza, że musimy czekać na rezultat działania (możemy założyć, że wyłącznie zlecamy wykonanie procedury, pobierając wynik potem),ale że zakładamy niezawodność komunikacji między urządzeniami.

Wiadomości Komunikacja za pomocą wiadomości jest asynchroniczna, dodatkowo dobry system przesyłania wiadomości gwarantuje jej dostarcze-

nie do adresata, lub informację o jej niedostarczeniu. Musimy pamiętać, że warunki w jakich możemy komunikować się z urządzeniem przenośnym są dalekie od doskonałych, a mimo to użytkownik oczekuje pełnej niezawodności systemu. To sprawia, że należy go tak zaprojektować, aby przerwy w komunikacji były jak najmniej odczuwalne. Systemy komunikacyjne takie jak RPC,RMI czy CORBA projektowane były dla warunków w których problemy komunikacyjne są sytuacją nadzwyczajną. Tymczasem w przypadku łączności bezprzewodowej utrata połączenia nie jest niczym czego nie moglibyśmy się spodziewać. W tym momencie idea przesyłania wiadomości wydaje się skutecznym rozwiązaniem tych problemów. Podczas tworzenia przykładowej implementacji napotkaliśmy jednak na problemy, które uniemożliwiły wykorzystanie go do integracji, która jest przedmiotem tej pracy.

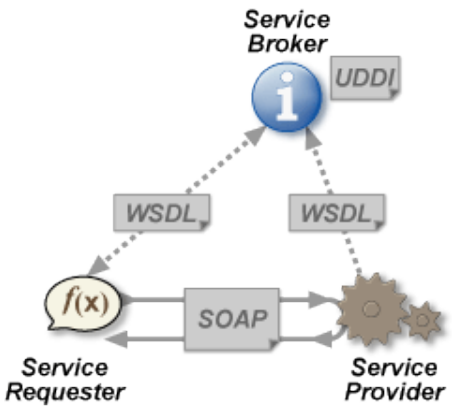
Web Services Technologia Web Service zsotała wykorzystana w przykładowej implementacji systemu integrującego, dlatego też jej szczegółowemu opisowi został poświęcony cały następny rozdział.

1.2 Zdalne wywoływanie procedur - Web Serwisy

Najpopularniejszą technologią związaną z zagadnieniem zdalnego wywoływania procedur jest technologia Web Service. W następnych rozdziałach przedstawiony zostanie zbiór ogólnych informacji o tej technologii oraz zostanie podjęta próba umiejscowienia jej w świecie rozwiązań mobilnych.

1.2.1 Podstawowe informacje o technologii Web Service

Pierwsze wzmianki o rozwiązaniu pozwalającym na współpracę różnorodnych systemów, poprzez ogólnodostępne medium pojawiły się w okolicach roku 1999. W krótkim czasie zainteresowały się nim wszystkie największe firmy działające na rynku rozwiązań informatycznych. U podstaw technologii Web Service leży Extensible Markup Language(XML). Na jego podstawie zbudowano dwa podstawowe języki towarzyszące rozwiązaniom typu Web Service : Standard Object Access Protocol (SOAP) oraz Web Services Definition Language (WSDL). Dodatkowo istnieje też technologia pozwalająca na stworzenie rejestru usług typu Web Service o nazwie Universal Description, Discovery, and Integration (UDDI). Relacje istniejące pomiędzy tymi trzema elementami przedstawia poniższy rysunek.



Standard Object Access Protocol

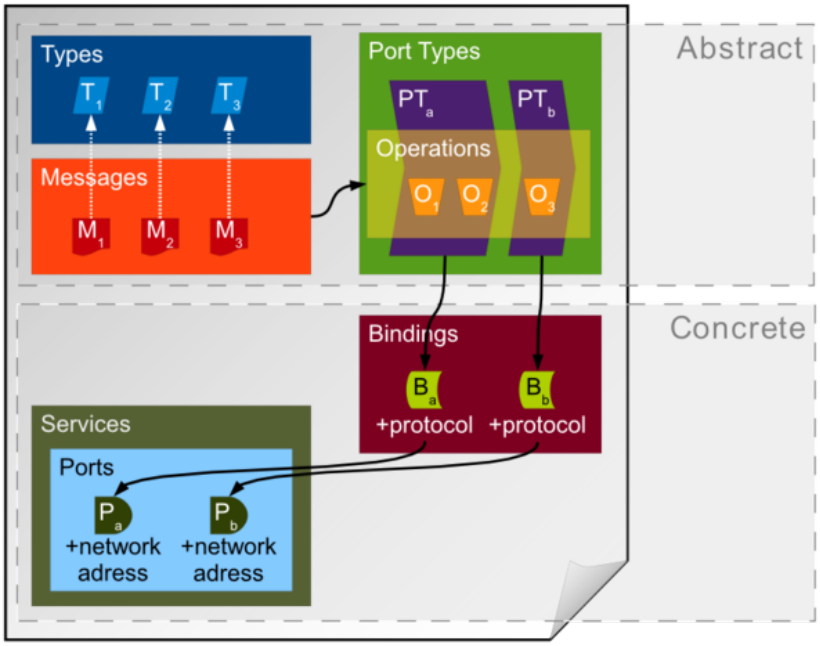
SOAP jest pewnym sposobem zapisu informacji posiadających strukturę. Wykorzystuje on język XML jako format, w którym zapisywane są wiadomości wymieniane pomiędzy poszczególnymi systemami komunikującymi się przy użyciu technologii Web Serivce. Sam protokół SOAP opiera swoje dzia-

łanie o inne protokoły warstwy aplikacyjnej , takie jak RPC czy też HTTP, które realizują za niego wszelkie negocjacje, czy też transmisje. SOAP stanowi podstawę dla stosu protokołu Web Service dostarczając elementarną funkcjonalność wymiany wiadomości, na której mogą być budowane wyższe, abstrakcyjne warstwy. Wykorzystanie tego protokołu jako podstawy Web Service niesie ze sobą dość poważne konsekwencje. SOAP będąc formą elektronicznej koperty generuje stosunkowo duży narzut danych na każdą wysyłaną wiadomość, przez co wpływa również na wydajność komunikacji przy użyciu Web Services. W związku ze wspomnianą powyżej wadą nie jest zalecane stosowanie tej technologii do zadań, w których kluczowy jest czas reakcji systemu. Ponieważ przedmiotem niniejszej pracy jest przedstawienie różnych metod integracji środowisk mobilnych, należy zwrócić uwagę na to, że może się zdarzyć, że pomimo wygody i standaryzacji jakiej dostarcza ta technologia, rozważenia wymaga to czy narzut wydajnościowy nie będzie stanowił przeszkody w rozwoju aplikacji. Szczególne znaczenie może mieć to w sytuacji, gdy potencjalny klient mobilnej aplikacji ma niezryczałtowany dostęp do sieci. Oznacza to, że narzut na rozmiar danych będzie się bezpośrednio przekładał na koszt wykorzystywania aplikacji. A więc będzie również wpływał na opłacalność wdrożenia danego rozwiązania mobilnego.

Web Service Description Language

WSDL jest to język oparty o XML, który umożliwia opisanie usługi typu Web Service. Używany jest często w połączeniu z protokołem SOAP oraz z XML Schema umożliwiając wystawienie technologii Web Service na zewnątrz, na przykład do internetu. Tak wystawiony opis usługi w postaci zestawu tagów WSDL pozwala aplikacji klienckiej na sprawdzenie jakie funkcje są dostępne na wskazanym serwerze. W przypadku aplikacji mobilnych dostęp do opisu

usługi Web Service najczęściej wykorzystywany jest przy tworzeniu oprogramowania metodą RAD (Rapid Application Development). W tym podejściu programista wskazuje miejsce gdzie znajduje się opis usługi, a następnie na jego podstawie generator aplikacji tworzy szkielet pozwalający podłączenie się do tej usługi oraz wywołanie wybranych funkcji. To podejście pozwala oszczędzić pracy potrzebnej na napisanie obsługi tego typu wywołań. Nie jest ono jednak dostępne dla wszystkich platform, gdyż generatory wymagają zwykle wsparcia po stronie urządzenia (na przykład potrzebują implementacji odpowiedniego profilu). By moć przejść do bardziej zaawansowanych zagadnień należy wyjaśnić sposób w jaki WSDL mapuje abstrakcyjne porty na konkretne usługi dostępne w systemie. WSDL definiuje usługę jako kolekcję punktów dostępowych lub też portów. Port definiowany jest poprzez asocjację adresu sieciowego z reużywalnym wiązaniem. Kolekcja portów definiuje usługę. Wiadomości stanowią abstrakcyjny opis wymienianych danych. Natomiast tak zwane Port Types stanowią abstrakcyjną kolekcję obsługiwanych operacji. Konkretny protokół oraz specyfikacja formatu danych dla wybranego portu tworzy reużywalne wiązanie, w którym operacje i wiadomości przywiązywane są do konkretnego protokołu sieciowego oraz do konkretnego formatu wiadomości. Schemat opisanych powyżej zależności został przedstawiony na poniższym rysunku.



Rodzaje usług Web Service

Pomimo, iż Web Services miało być jednolitym i ustandaryzowanym podejściem umożliwiającym łatwą wymianę dowolnych danych pomiędzy systemami różnego typu, to w ostatecznym rozliczeniu okazuje się, że nie dokońca jest to ujednolicone. Przy ustalaniu powiązania usługi z protokołem komunikacyjnym mamy do dyspozycji dwa style wiązania :

- Styl Remote Procedure Call (RPC)
- Styl dokumentowy

Co więcej samo wiązanie SOAP może posiadać dwa sposoby użycia :

- Zakodowany (encoded)

- Dosłowny (literal)

To daje nam razem cztery możliwe modele :

- RPC/encoded
- RPC/literal
- Document/encoded
- Document/literal

Oprócz wyżej wymienionych wyróżnia się jeszcze piąty model związany ze stylem wiązania zwany document/literal wrapped. Wiedza o istnieniu tego typu podziału jest kluczowa do poprawnej implementacji konsumenta Web Service. W związku z ograniczonymi zasobami zarówno pamięciowymi jak i wydajnościowymi konsumenci pisani na urządzenia mobilne muszą być od razu konfigurowani pod kątem obsługi konkretnego modelu. Nie są w stanie dokonać analizy dokumentu WSDL generowanego po stronie serwera. Co więcej należy bardzo uważać na to jakie style wiązań jest w stanie obsłużyć zarówno klient jak i serwer. Często każda ze stron potrafi poradzić sobie tylko z niektórymi ich rodzajami. W skrajnym przypadku dochodzi także do różnic pomiędzy wersjami. Taka sytuacja zachodzi na przykład w Apache Axis, który wykorzystywany jest przy implementacji przykładowego systemu integracyjnego.

RPC/encoded

Najbardziej opisowy styl wiązania. Wyróżnia się dodawaniem do argumentów zapytania dodatkowych informacji o typie (na przykład xsd:int w poniższym przykładzie).

```
<soap:envelope>
    <soap:body>
        <myMethod>
            <x xsi:type="xsd:int">5</x>
            <y xsi:type="xsd:float">5.0</y>
        </myMethod>
    </soap:body>
</soap:envelope>
```

RPC/literal

W przypadku stylu RPC/literal zredukowano argumenty wywołania do samych wartości, pozbywając się redundantnych danych.

```
<soap:envelope>
    <soap:body>
        <myMethod>
            <x>5</x>
            <y>5.0</y>
        </myMethod>
    </soap:body>
</soap:envelope>
```

Document/literal

Obecnie najsilniej wspierany i promowany model. Jego największą zaletą jest to, że może podlegać walidacji przy użyciu XML Scheme. Jak widać w poniższym przykładzie w kodzie wywołania brakuje nazwy metody. Jest ona przekazywana w nagłówku zapytania HTTP.

```
<soap:envelope>
    <soap:body>
        <xElement>5</xElement>
        <yElement>5.0</yElement>
    </soap:body>
</soap:envelope>
```

Jest to najbardziej interesujący model z punktu widzenia niniejszej pracy. Gwarantuje on stosunkowo najmniejszy narzut oraz jest nieznacznie szybszy od pozostałych. Co więcej jest wspierany przez wszystkie najnowsze wersje konsumentów Web Services.

1.2.2 Bezpieczeństwo a Web Service

Z myśla o bezpiecznnej komunikacji przy użyciu technologii Web Service wprowadzono protokół komunikacyjny WS-Security (Web Service Security), pozwalający na zaaplikowanie podstawowych standardów bezpieczeństwa. Protokół ten zawiera specyfikację opisującą metody wymuszania integralności oraz poufności przesyłanych danych. Pozwala na dołączanie do wiadomości SOAP podpisów oraz nagłówków związanych z szyfrowaniem danych. Wspiera taże formaty certyfikacyjne takie jak X.509. Protokół działa w warstwie aplikacyjnej i został tak zaprojektowany by zapewnić bezpieczeństwo od końca do końca (end-to-end security). Niestety w obecnej chwili protokół ten nie został zaimplementowany na urządzeniach mobilnych i na pewno w najbliższym czasie nie stanie się na nich standardem. Ograniczeniem tu są typowe dla tych urządzeń problemy z wydajnością. Alternatywnym rozwiązaniem problemu bezpieczeństwa jest posłużenie się zabezpieczeniami na poziomie warstwy transportowej (TLS) takimi jak https. Pozwala to na zapewnienie bezpieczństwa punkt-punkt.

1.3 Systemy oparte na przesyłaniu wiadomości

Systemy przesyłania wiadomości pozwalają na integrację systemów z uwzględnieniem specyfiki każdego z nich. Jednak jak każde rozwiązanie mają swoje mocne i słabe strony.

Niezależność integrowanych systemów Komunikacja między systemami jest asynchroniczna, co z jednej strony powoduje, że system musi obsługiwać bardziej skomplikowane scenariusze w których odpowiedź na żądanie może nie pojawić się od razu, z drugiej pozwala na działanie komponentom względnie niezależnie do czasu odzyskania komunikacji. W razie awarii jednego z systemów, drugi może cały czas wysyłać wiadomości do kanału komunikacyjnego, które zostaną odebrane po odzyskaniu sprawności przez odbiorcę.

Routing Wiadomości mogą pokonywać skomplikowaną drogę zanim dotrą do miejsca przeznaczenia. System wysyłający wiadomość nie musi wiedzieć jak dostarczyć ją do odbiorcy, jedyne co musi zrobić, to wstawić ją do kolejki. Daje to duże możliwości zmian w konfiguracji systemu, bez ingerowania w aplikację.

Transformacja System przesyłania wiadomości może dokonywać zmian w wiadomościach zgodnie z ustalonymi regułami. Pozwala to na dalsze uniezależnienie integrowanych systemów od siebie: każdy system może wysyłać i odbierać wiadomości we właściwym dla siebie, natywnym formacie, podczas gdy za tranformację odpowiada kanał komunikacyjny.

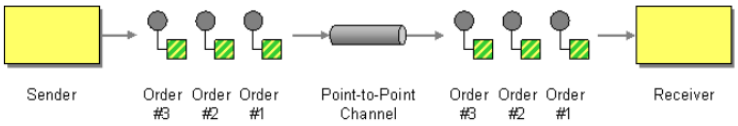
1.3.1 Czas reakcji

Przesyłanie i przetwarzanie wiadomości ustępuje wydajnością integracji na poziomie danych, gdzie nie pojawia się narzut związany z wyłuskiwaniem danych i opakowywaniem ich do ustalonego formatu. Istnieje jednak wiele zastosowań, gdzie szybkość nie gra najważniejszej roli,a przesyłane są jedynie niewielkie (ale częste) porcje informacji. Nawet sytuacje w których należy przesłać ogromną ilość danych, np. podczas migracji systemów mogą zostać obsłużone w procesach batchowych, wykonywanych poza godzinami normalnego użytkowania systemu.

Sposoby połączenia

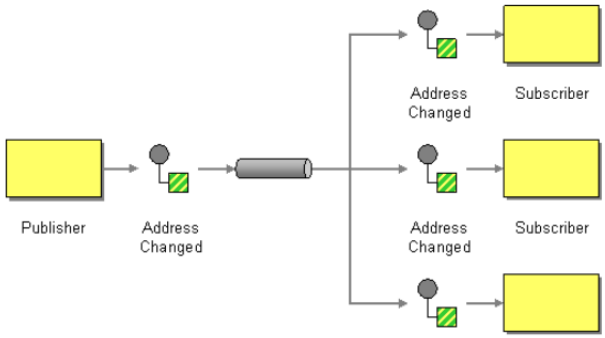
W systemach komunikujących się za pomocą wiadomości można wyróżnić dwa różne sposoby połączenia komponentów.

Kanał Point-to-point



W tym przypadku wiele procesów może być podłączonych do obu stron kanału komunikacyjnego. Z jednej strony kilka równolegle działających procesów może wysyłać wiadomości, z drugiej strony procesy mogą je odbierać. Ponieważ jednak równoległe przetwarzanie tej samej wiadomości przez kilka procesów mogłoby być nie porządane, kanał komunikacyjny dba o to by jedna wiadomość mogła być pobrana tylko przez jeden z nich. Pozwala to na znakomite zrównoleglenie przetwarzania wiadomości przez proste dodawanie odbiorców i podłączanie ich do kanału komunikacyjnego.

Kanał Publish-Subscribe



Jeżeli chcemy udostępnić pewne informacje szerszej grupie procesów możemy utworzyć kanał typu Publish-Subscribe. W tym przypadku procesy re-

jestrują się w menedżerze kolejki wybierając temat (Topic) z którego chciałyby otrzymywać wiadomości. Zadaniem menedżera kolejki jest zapewnienie by wszystkie zarejestrowane procesy otrzymały wstawioną do kolejki wiadomość, oczywiście każdy z nich jeden raz.

Java Message Service Java Message Service jest standardem oferującym aplikacjom Javowym możliwość tworzenia, wysyłania, odbierania i czytania wiadomości. JMS jest jedynie specyfikacją, pozwalającą na jednolite korzystanie z różnych systemów przesyłania wiadomości. Porównując konkretną implementację z bazą danych, można powiedzieć, że JMS oferuje podobne zunifikowane API do przesyłania wiadomości, jak JDBC zunifikowany interfejs dostępu do bazy danych. Istnieje wiele implementacji JMS, zarówno komercyjnych jak i dostępnych bezpłatnie. Istnieją wersje stand-alone, jak np IBM WebsphereMQ czy Apache ActiveMQ, jak również systemy przesyłania wiadomości wbudowane w serwery aplikacyjne. Niestety nie wszystkie pozwalają na wymianę wiadomości z urządzeniami mobilnymi.

Rozdział 2

Mechanizmy integracyjne w technologiach mobilnych

2.1 Szybka droga do mobilności ?

Na wstępie dyskusji o mobilnej części platformy integracyjnej zwrócimy uwagę na dostępne na rynku urządzenia, które posiadają preinstalowane systemy operacyjne pozwalające na dodawanie nie przewidzianych przez twórców, zewnętrznych rozwiązań. Tego typu systemy operacyjne muszą się charakteryzować udostępnieniem na zewnątrz, do instalowanych przez nas aplikacji pewnego API pozwalającego na realizację podstawowych usług takich jak transfer danych czy ich składowanie w pamięci urządzenia. Na chwile obecną (drugi kwartał 2008 roku) na rynku dostępne są urządzenia działające w oparciu wymienione poniżej platformy mobilne :

- WMD - Windows Media Devices oparte o system operacyjny Windows Mobile opracowany przez firmę Microsoft

- Symbian - platforma spotykana głównie na smartfonach oferowanych przez firmy takie jak Nokia czy Samsung
- Blackberry - marka wypromowana prze firmę Research in Motion, oferuje urządzenia z preistalowanym systemem operacyjnym opartym na wirtualnej maszynie Java
- Android - system operacyjny oparty na Linuxie, możliwy do rozbudowywania głównie przez aplikacje J2ME (w obecnej chwili system jest w fazie testów)
- pozostałe autorskie systemy operacyjne, udostepniające standardową maszynę wirtualną Java

Po dokładnym przyjrzeniu się powyższym platformom nasuwa się jeden stanowczy wniosek - poza platformami opartymi na J2ME, wykonanie jednej, spójnej aplikacji spełniającej założenia interoperacyjności jest praktycznie niemożliwe. Jedynym rozwiązaniem jest przygotowanie kilku instancji tej samej aplikacji, które niestety nie będą miały ze sobą niczego wspólnego poza przyjętymi założeniami o podstawowej funkcjonalności. Wynika to między innymi z tego, że każda platforma oferuje inne możliwości i przy okazji wprowadza inne ograniczenia. Co więcej tego typu różnicowanie występuje również w obrębie samych platform. Na przykład J2ME posiada profile związane na stałe z modelami urządzeń, które decydują o tym, jakie funkcjonalności dostępne są dla programów. Wskazane powyżej ograniczenia techniczne wymuszają przy próbie wejścia na rynek aplikacji mobilnych,już na etapie planowania produktu, wybranie konkretnej platformy docelowej na jakiej oferowane będzie nasze rozwiązanie. Wraz z sukcesem rozwiązania, możliwa będzie dalsza ekspansja na pozostałe platformy. Przed tą ostateczną ewaluacją wartości pomysłu na produkt, jest bardzo ryzykownym podjęcie próby równoległego

wprowadzenia go na różnych, niekompatybilnych platformach. Zwiększa to znacznie próg wejścia oraz utrudnia poprawną kontrolę nad stopniem dojrzałości produktu. Co więcej przy słabym zarządzaniu funkcjonalnościami może wystąpić problem zróżnicowania pomiędzy różnymi wersjami naszej aplikacji. Tego typu problemy wpłyną w sposób bezpośredni na postrzeganie naszego produktu przez klientów, w szczególności gdy będziemy mieli do czynienia z korporacjami, w obrębie których stosowane są rózne rozwiązania mobilne. W wypadku naszego systemu integracyjnego naturalnym wyborem będzie J2ME. Na chwilę obecną jest jedynym standardem, który daje jakiekolwiek nadzieje na uzyskanie dominacji na rynku. Co więcej platforma Blackberry, która jest stworzona właśnie w oparciu o ten standard jest jedną z najbardziej popularnych, w firmach, które chcą podjąć działania mające na celu umobilnienie, występujących w ich obrębie, procesów biznesowych.

2.2 Jak się do tego zabrać ?

W momencie gdy mamy ustaloną już platformę docelową (J2ME) musimy przyjrzeć się temu co nam oferuje. Z punktu widzenia integracji najważniejsze dla nas są dwie cechy :

- możliwość do trwałego przechowywania danych,
- zdolności do komunikacji z systemami zewnętrznymi.

Pod pojęciem trwałego przechowywania danych kryje się niewrażliwość danych na przerwy w dostawie energii do urządzenia lub ewentualne przypadkowe restarty. Platforma J2ME oferuje nam tu standardowe rozwiązanie w postaci Persistance API dające dostęp do tzw. Record Store-ów pozwalających zapisywać dowolne dane w pamięci trwałej urządzenia. Każde urządzenie posiadające maszynę wirtualną java musi obowiązkowo posiadać tego

typu pamięć. API to jest bardzo prymitywne i nie pozwala na przechowywanie danych w sposób relacyjny, czy nawet nie pozwala na serializowanie obiektów. Jednak przy zastosowaniu odpowiedniego poziomu abstrakcji, poprzez opakowanie tych interfejsów w odpowiednie klasy jesteśmy w stanie uzyskać wygodny sposób na przechowywanie danych. Wymaga to niestety sporo dodatkowej pracy. Przy analizowaniu zdolności do komunikacji sprawy się znacznie komplikują. Istnieje kilka, dość znacząco różnych metod komunikacji, które możemy wykorzystać do synchronizacji danych z naszymi zewnętrznymi źródłami. U podstaw komunikacji leżą takie protokoły takie jak Direct TCP czy Http, które pozwalają na wymianę czystych nie przetworzonych danych binarnych, czy też tekstowych. Można je wykorzystać w sposób bezpośredni, jednak jest to niezwykle trudne i niewygodne. Z pomocą przychodzą nam protokoły komunikacyjne wyższego poziomu. Ich wadą jest to, że często nie są elementem standardowego wyposażenia urządzeń, na które będziemy pisali naszą aplikację. Nie jest to jednak problemem, ze względu na to, że u podstaw tychże protokołów wyższego poziomu zawsze leży protokół podstawowy, który jest dostępny bezpośrednio w standardowej bibliotece. Co więcej istnieją dziesiątki darmowych implementacji protokołów wyższego poziomu, które przy zerowym niemal koszcie, mogą zostać zaadoptowane do naszych potrzeb. Poniżej przedstawimy krótki przegląd rozwiązań wysokiego poziomu, które pozwalają na wygodną i łatwą do prześledzenia komunikację z systemami zewnętrznymi.

Istnieją cztery podstawowe sposoby, na które można zorganizować wysokopoziomową komunikację pomiędzy urządzeniem mobilnym a źródłem danych :

- Protokół binarny - najbardziej skomplikowana forma komunikacji, wymaga zaprojektowania własnego protokołu, a więc jest to metoda naj-

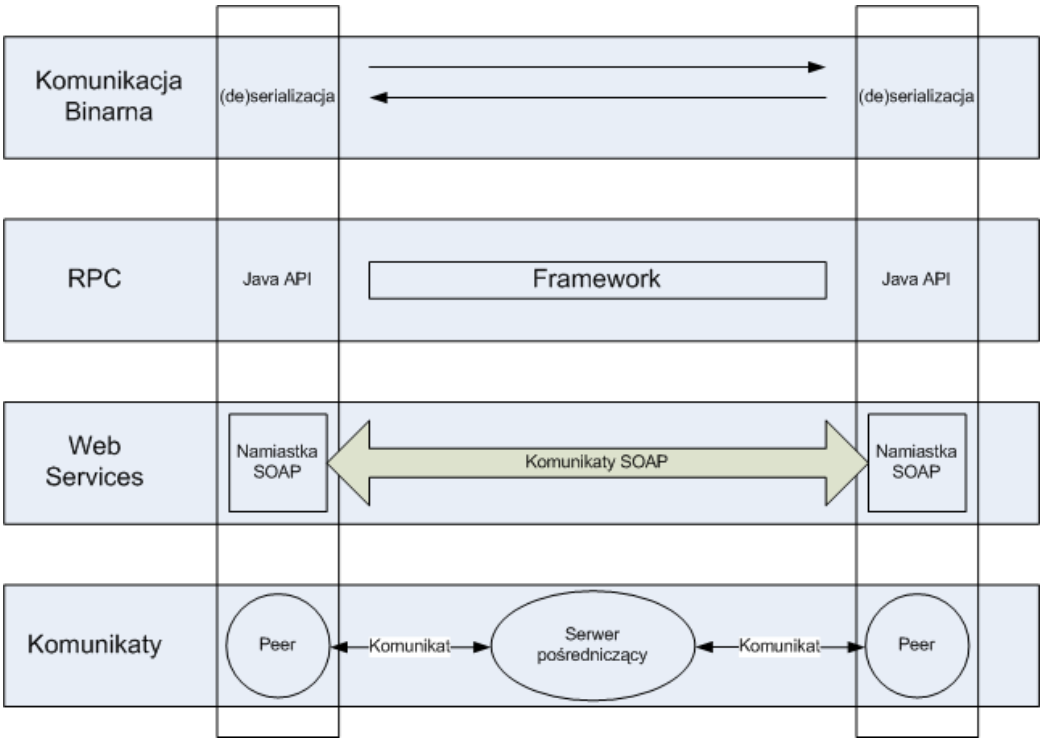
bardziej podatna na błedy programisty/projektanta.

- RPC - wykorzystanie narzędzi do tworzenia standardowej komunikacji opartej o zdalne wywoływanie procedur, dużo prostsze i odporniejsze na błedy niż poprzednia metoda, jednak wymaga bardzo bliskiego związania serwera z klientem, co zmniejsza interoperacyjność rozwiązania.

- Web Services - bardzo popularna metoda pozwalająca na zbudowanie komunikacji pomiędzy dowolnymi systemami w oparciu o standaryzowane, niezależne od końcowych systemów komunikaty XML. Budowa aplikacji w oparciu o Web Servces jest bardzo prosta i częściowo zautomatyzowana, jeśli zastosujemy odpowiednie frameworki. Największa zaletą jest tu całkowita interoperacyjność. Rozwiązanie to posiada niestety również wady w postaci zwiększonego ruchu w sieci oraz zwiększenia wymagań wobec sprzętu, na którym je wykorzystujemy.

- JMS - wprowadzenie elementu pośredniczącego pomiędzy urządzeniem a źródłem danych. Komunikacja polega na przesyłaniu wiadomości, które w razie awarii jednego z sytemów mogą zostać przechowane na kolejce komunikatów, która znajduje się na systemie pośredniczącym w komunikacji.

Z powyższego zestawinia można wyróżnić dwie technologie - Web Services oraz JMS. Obie pozwalają na wysoką interoperacyjność rozwiązania, co w sytuacji gdy mamy przed sobą perspektywę implementacji naszego rozwiązania na dziesiątkach typów urządzeń jest kluczowym zagadnieniem do rozpatrzenia. Wybór pomiędzy nimi sprowadza się do ustalenia czy niezbędna dla naszego projektu jest obecność systemu pośredniczącego, który jest w stanie zwiększyć niezawodność komunikacji. Czynnikiem decyducjącym tu jest ilość źródeł danych oraz sposób ich wykorzystywania przez użytkowników. Jeśli



Metody komunikacji

okazałoby się, że przez system będzie przechodziło dużo komunikatów, które będa mogły doprowadzić do przeciążenia systemów źródłowych, element pośredniczący buforujący komunikaty może okazać się niezbędny. Jednak w wypadku gdy każdy użytkownik będzie korzystał z danych, które dostępne są przez niezależne kanały - czyli systemy źródłowe nie będą posiadały wąskiego gardła, należałoby rozważyć skorzystanie z web services, które są znacznie tańszym rozwiązaniem niż JMS. Wynika to z tego, że większość implementacji WS dostępna jest za darmo w postaci Open Source lub jest już wbudowana w systemy źródłowe, a co więcej nie wymaga dodatkowego serwera buforującego komunikaty.

Rozdział 3

Szablon komunikacyjny

Do zademonstrowania możliwości komunikacyjnych pomiędzy systemem stacjonarnym i mobilnym stworzyliśmy szablon ułatwiający tworzenie aplikacji posiadających mobilny interfejs, z poziomu którego można sterować aplikacją serwerową. Rozwiązanie, które przyjęliśmy jest na tyle ogólne, że umożliwia rozwiązanie kilku podobnych problemów, które przedstawimy dalej.

3.1 Istniejące sposoby dostępu do informacji z urządzenia mobilnego

Przyjrzyjmy się jednak najpierw możliwym sposobom zdalnego dostępu do danych z poziomu urządzenia mobilnego. Przedstawione sposoby być może nie wyczerpują wszystkich możliwości podziału, jednak dają obraz tego w jaki sposób tworzone są dzisiaj aplikacje mobilne.

Mobilne przeglądarki Urządzenia mobilne, w tym telefony komórkowe, mają wbudowane przeglądarki internetowe, za pomocą których coraz częściej można przeglądać wszystkie dostępne w Internecie strony (w przeciwieństwie

do sytuacji sprzed kilku lat, gdy jedynie część z nich, posiadająca wersję WAP, była dostępna). Trzeba przyznać, że w tym czasie dokonał się duży postęp w rozwoju mobilnych przeglądarek. Ciągle jednak nie zapewniają one wygody porównywalnej z przeglądaniem stron na tradycyjnym komputerze. Z drugiej strony, korzystanie z aplikacji zbudowanych w oparciu o model trójwarstwowy (z użyciem tzw. cienkiego klienta - czyli przeglądarki WWW) niesie za sobą szereg korzyści, jak brak potrzeby instalowania aplikacji,łatwość wdrożenia czy wreszcie całkowita przenośność.

Aplikacje mobilne Innym podejściem, przypominającym aplikacje zbudowane w oparciu o model dwuwarstwowy, jest stworzenie aplikacji mobilnej, która samodzielnie łączy się z Internetem i umożliwia wyświetlanie i modyfikację danych w sposób przewidziany przez jej twórców. Ogranicza to oczywiście funkcjonalność takiej aplikacji do zakresu przewidzianego w danej wersji, a wszelkie zmiany wymagają jej aktualizacji. W zamian za to otrzymujemy większą szybkość działania, możliwość walidacji danych po stronie klienta oraz elastyczność w tworzeniu interfejsu (który w tym przypadku nie jest ograniczony do kontrolek zapewnianych przez przeglądarkę WWW).

Wadą, z której niewiele osób zdaje sobie sprawę, jest mniejsza przenośność aplikacji, choćby były one napisane w języku projektowanym z myślą o niezależności od platformy, takim jak Java. Wynika ona z różnorodności urządzeń dostępnych na rynku. Całkowita przenośność możliwa jest do uzyskania tylko dla prostych aplikacji, nie korzystających z możliwości oferowanych przez konkretne modele urządzeń mobilnych. Przy próbie zbudowania czegokolwiek większego natychmiast natkniemy się na niewidzialny mur generowany przez ogromne różnice pomiędzy dostępnością profili na poszczególnych platformach oraz przez różnice w implementacji pewnych części samych maszyn

wirtualnych. Co więcej tu opisana sytuacja wskazuje na problem, który generuje stworzona z myśla o przenośności platforma J2ME. Jeszcze większe problemy pojawiają się gdy chcemy rozszerzyć zasięg naszej aplikacji o kolejne segmenty rynku, takie jak użytkownicy systemów Symbian czy Windows Mobile. W tym momencie okazuje się, że zostaniemy zmuszeni do napisanie nie jednej aplikacji, ale wielu, które często mogą nie mieć prawie żadnych części wspólnych.

Generatory aplikacji Wielu producentów podejmuje próby zbudowania środowisk umożliwiających szybkie tworzenie aplikacji mobilnych (tak zwane RAD - Rapid Application Development). Mają one łączyć zalety obu przedstawionych wcześniej światów :

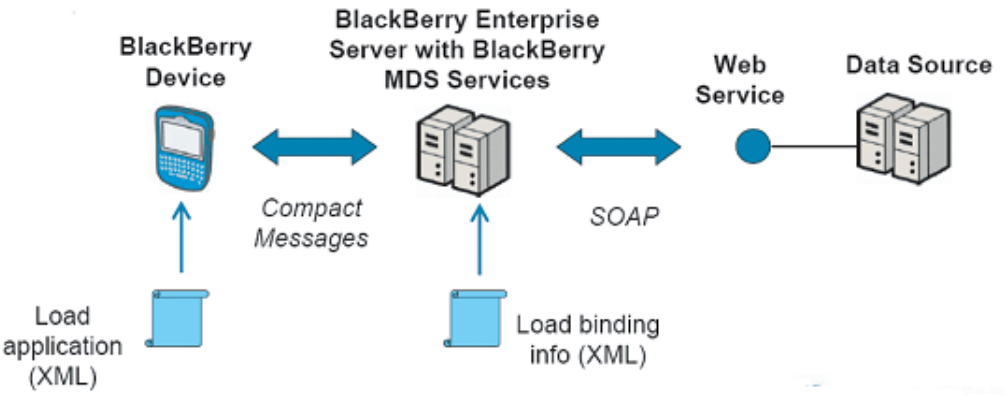
- Pozwalać na łatwe i szybkie tworzenie aplikacji dostępowych - tak jak mobilne przeglądarki
- Dostarczać elastyczności i rozbudowanych interfejsów użytkownika - tak jak aplikacje mobilne

W dalszej części niniejszej pracy przedstawione zostaną dwa tego typu rozwiązania, pozwalające na oszczędzenie znacznej ilości pracy przy budowaniu nowych aplikacji.

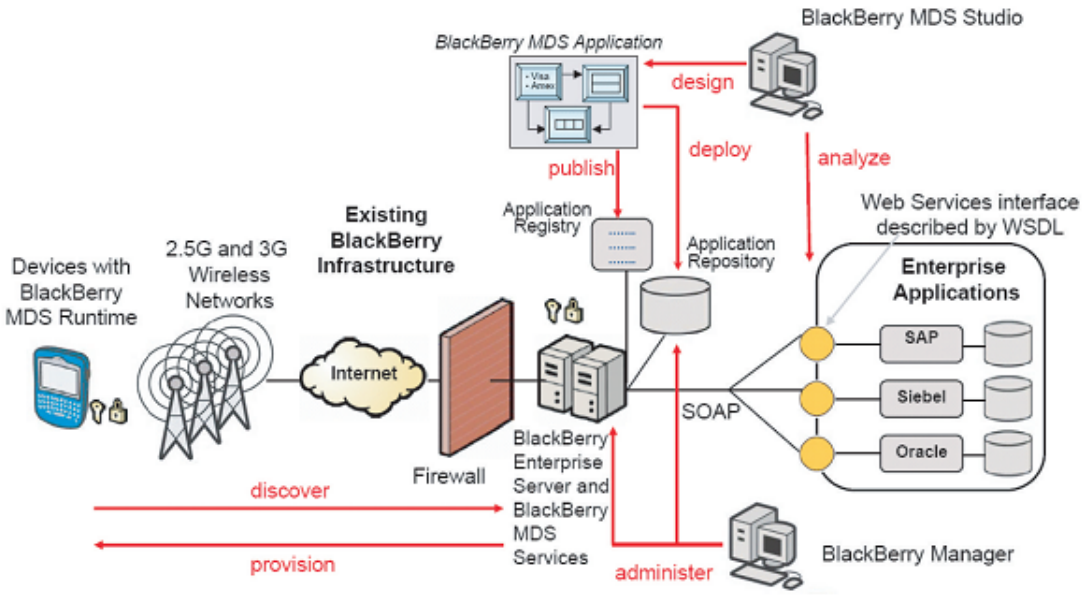
Blackberry MDS

Blackberry MDS jest przykładem narzędzia wyspecjalizowanego do tworzenia aplikacji tylko i wyłącznie na platformę jednego producenta. Rozwiązanie to opiera się na integracji małej aplikacji zainstalowanej na urządzeniach klienckich (MDS Runtime) z usługą typu Web Service działającą na serwerze. Dzięki niemu możliwe jest tworzenie aplikacji używając podejścia opisowego.

Zamiast pisać całą aplikację używając języka programowania wraz z bibliotekami oferującymi interfejs użytkownika, możemy przy użyciu graficznego edytora projektować takie elementy jak ekrany czy pola z danymi. Wszystko to przy użycia standardowego drag and drop. Następnie możemy połączyć tak zbudowane komponenty przy użyciu różnych przewodników(wizards) czy też edytorów. Co więcej aplikacja MDS daje nam pełną kontrolę nad przepływem sterowania - przy pomocy języka JavaScript jesteśmy w stanie zaimplementować niemal dowolną logikę aplikacji. Aplikacje typu enterprise oraz źródłą danych dostępne są dla aplikacji mobilnej poprzez standardową technologię Web Service.



Aplikacja MDS dla Blackberry jest to aplikacja typu rich-client zbudowana na podstawie metadanych zdefiniowanych w języku XML. Te metadane wraz z zasobami takimi jak obrazki gromadzone są w pakietach, które następnie publikowane są na serwerze BES. Serwer ten przy użyciu technologii push rozsyła aplikacje do terminali mobilnych Blackberry. Dzięki zcentralizowanemu sposobowi dystrybucji oraz zarządzania aplikacjami mobilnymi koszty utrzymania mobilnej infrastruktury są znacznie niższe.



Usługa Blackberry MDS Usługa MDS Blackberry została zaprojektowana tak by umożliwić wymianę danych pomiędzy urządzeniami Blackberry a aplikacjami typu enterprise. Zapewnia ona :

- Izolację platformy od szczegółów związanych z integracją źródeł danych
- Standard asynchronicznej wymiany komunikatów
- Przezroczyste dla użytkownika i developera protokoły bezpieczeństwa
- Transformacje wiadomości
- Wspomnianą wcześniej zcentralizowaną administrację przy pomocy technologii push

Blackberry MDS runtime Po stronie urządzenia mobilnego zainstalowany jest kontener aplikacji MDS. Zajmuje się on instalacją, wyszukiwanie

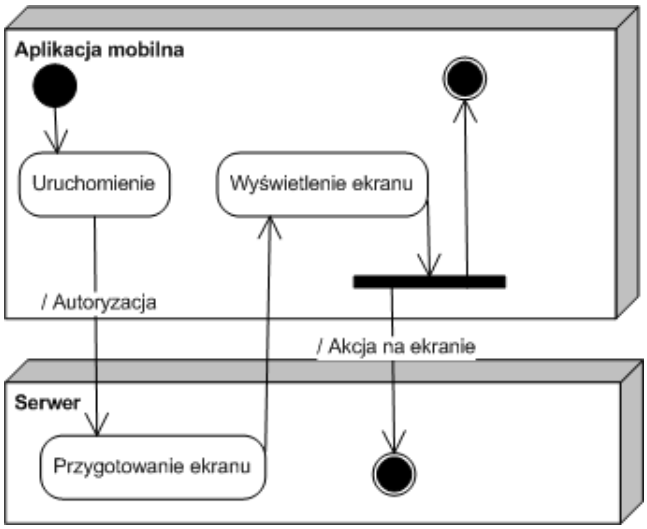
oraz przeglądaniem aplikacji MDS. Dostarcza również niezbędne funkcjonalności do tych aplikacji, takie jak interfejs użytkownika, kontener danych oraz usługi komunikacyjne klient-server.

Netbeans Mobile IDE

3.2 Idea szablonu

Ideą stojącą za stworzeniem szablonu było połączenie zalet zapewnianych przez aplikację zbudowaną w oparciu o model trójwarstwowy z tymi typowymi dla samodzielnej aplikacji. Stworzone rozwiązanie stanowi raczej pomysł, który można rozwijać i wykorzystywać do budowy bardziej skomplikowanych systemów. Dlatego też możliwości takiego podejścia nie są ograniczone przez stworzony szablon.
Szablon składa się z dwóch części - mobilnej i serwerowej. Użytkownik może określić wygląd ekranu przesyłanego do urządzenia mobilnego, który po stronie mobilnej interpretowany jest do postaci elementów interfejsu ciężkiego klienta, na przykład pól tekstowych, wyboru czy rozwijanych list. Rozszerza to możliwości aplikacji w stosunku do zwykłej strony internetowej o elementy niedostępne w przypadku statycznej strony WWW jak rysunki wektorowe,obsługa przycisków urządzenia czy możliwość wyświetlania strony przez jakiś czas.

W typowym przypadku scenariusz użycia aplikacji będzie wyglądał następująco:



- Tworzymy nowy 'ekran' na serwerze. Oznacza to określenie wyglądu i zachowania aplikacji w formacie XML. Określamy dostępne dla danego ekranu akcje.

- Aplikacja mobilna łączy się z serwerem i pobiera przygotowany dla niej ekran. Użytkownik wykonuje jedną z dostępnych na nim akcji, a informacja o tym jest przesyłana do serwera. Dodatkowo aplikacja może wykonać dowolne zdefiniowane akcje. W ten sposób pojawia się możliwość kontrolowania ekranu po stronie klienta w dowolny sposób.

- Aplikacja mobilna pobiera kolejny ekran przygotowany przez serwer i dalej postępuje według powyższego schematu.

3.2.1 Możliwości

Aplikacja oparta o tak zaprojektowany model posiadałaby cechy typowe dla modelu dwuwarstwowego - w tym szybki czas reakcji, niezależność wyglądu aplikacji od stosowanego klienta oraz możliwość przeprowadzenia pewnych dodatkowych operacji (jak walidacja) po stronie klienta. W idealnym

przypadku mógłby być dynamicznie przesyłany kod wykonywany po stronie klienta (odpowiednik JavaScript w przypadku przeglądarki internetowej). Pozbawiona byłaby też typowych wad obu rozwiązań - brak konieczności aktualizowania aplikacji przy drobnych poprawkach (poza błędami w zainstalowanym oprogramowaniu). Nie byłaby również ograniczona do mechanizmów oferowanych przez statyczne strony WWW - można by było wyświetlać dowolne ekrany, a na nich wykonywać dowolne akcje. Dodatkowo, ponieważ stan aplikacji przechowywany byłby na serwerze, można w dowolnym momencie zmienić urządzenie mobilne, wyłączyć aplikację, a potem powrócić do zapamiętanego wcześniej miejsca.

Nowe podejście W przypadku tworzenia aplikacji mobilnych naturalna jest chęć przeniesienia funkcjonalności dostępnych w aplikacjach 'biurkowych' na urządzenie mobilne. Nie zawsze jest to podejście słuszne - użytkownik korzystający z urządzenia mobilngo - palmtopa czy telefonu komórkowego ma znacznie ograniczone możliwości działania i tym co się w takiej sytuacji liczy jest szybkość i łatwość obsługi aplikacji, którą można postawić przed jej dużymi możliwościami. Dlatego naszym zdaniem, celem tworzenia aplikacji mobilnych nie powinno być dostarczenie wszystkich możliwych funkcjonalności, ale raczej maksymalna ergonomia i prostota użycia, które przyświecały nam w trakcie tworzenia szablonu.

Ponadto, techniczne rozwiązania przyjęte w szablonie nie były - zgodnie z naszą wiedzą - do tej pory stosowane. Nasz szablon opiera się na bibliotece KUIX, która pozwala na wykorzystanie formatu XML do stworzenia interfejsu użytkownika. Pozwala to na dynamiczne ładowanie komponentów wcześniej przygotowywanych na serwerze. Naturalną drogą rozwoju byłoby rozbudowanie szablonu o możliwość dynamicznego ładowania kodu, mecha-

nizmów walidacji czy tworzenia grafiki.

3.2.2 Możliwe zastosowania

Opisywana idea ułatwia tworzenie określonego typu aplikacji i tak jak każdy szablon nie pokrywa wszystkich istniejących rozwiązań. Nasz szablon ułatwi pisanie aplikacji, które nie wymagają przesyłania dużych ilości danych i od której oczekujemy interaktywności większej niż w przypadku strony internetowej pozbawionej JavaScript.

Ankietowanie Jednym z zastosowań które możemy sobie wyobrazić jest zdalne ankietowanie użytkowników. Stworzenie aplikacji która na ekranie wyświetla listę dostępnych odpowiedzi i pozwala na wybranie jednej z nich, jest przy użyciu przedstawianego szablonu prostym zadaniem.

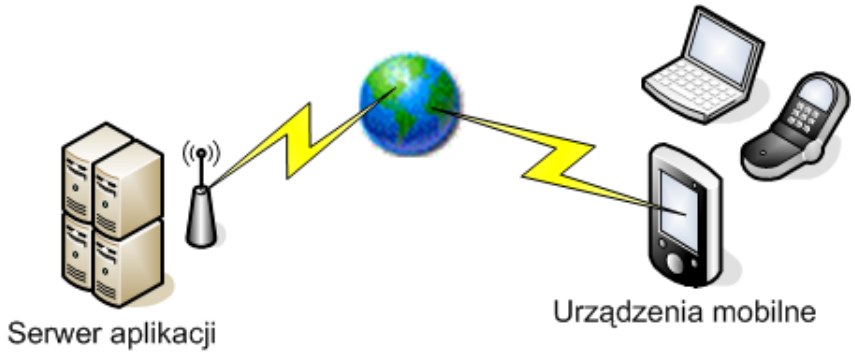
Zdalne podejmowanie decyzji Podobnie jak powyżej, użytkownicy mogą wybrać jedną z możliwych decyzji, w takim przypadku potrzebne będzie rozbudowanie szablonu o zaawansowane mechanizmy bezpieczeństwa, jednak idea pozostaje podobna.

Zdalne wyświetlanie treści Urządzenie mobilne może być też jedynie miejscem które wyświetla dostarczane do niego informacje. Interakacja z użytkownikiem może być w tym momencie wyłączona, a urządzenie mobilne może pełnić rolę zdalnie kontrolowanego ekranu.

3.2.3 Architektura

Tak jak zostało to już wspomniane szablon składa się z dwóch głównych części - serwerowej oraz mobilnej. Przed przejściem do szczegółów implemen-

tacyjnych zaprezentujemy koncepcję całego systemu.



Jak widać, urządzenia mobilne będą się łączyły przez Internet do uruchomionego serwera aplikacyjnego. Każde z nich wysyła żądanie pobrania ekranu, który jest dynamicznie przygotowywany w zależności od stanu aplikacji dla danego użytkownika. Warto tu zwrócić uwagę, że szablon jest pomyślany właściwie wyłącznie dla urządzeń mobilnych - stosowane rozwiązania, są albo niedostępne, albo mało przydatne w przypadku komputerów stacjonarnych, o czym napiszemy w dalszej części pracy.
Podczas projektowania szablonu pojawiły się wątpliwości co do sposobu komunikacji pomiędzy elementami systemu. Początkowa i bardzo przekonująca koncepcja zakładała wykorzystanie JMS - Java Messaging System, specyfikacji umożliwiającej przesyłanie wiadomości w Javie. Przemawiał za tym szereg argumentów:

- Niezawodność - gwarancja dostarczenia wiadomości w warunkach łączności bezprzewodowej, która jest szczególnie narażona na przerwy w transmisji.

- Niezależność systemów - użycie wiadomości znacznie ułatwiłoby ewentualne zmiany po stronie serwera aplikacyjnego, ze względu na dodatkowy element pośredniczący jakim jest serwer wiadomości.

- Łatwość implementacji - przesyłane wiadomości mogą zawierać zserializowane obiekty, co zdejmuje z programisty obowiązek samodzielnej serializacji i deserializacji danych.

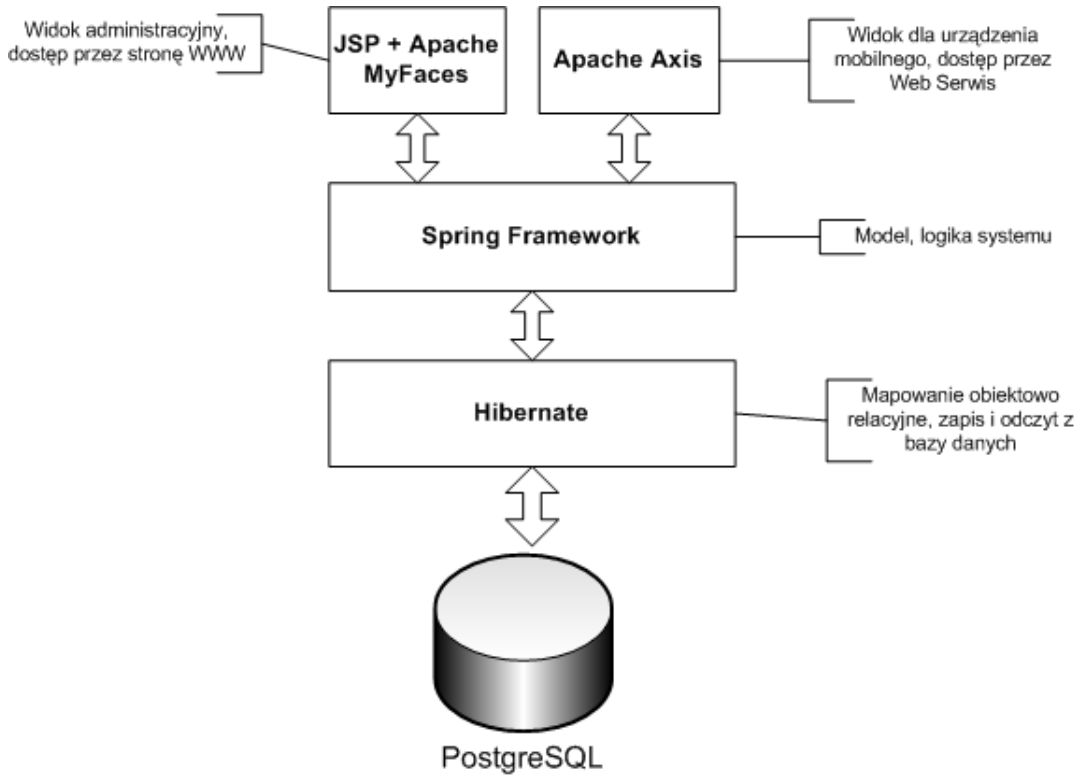
Jak się jednak okazało,istotne ograniczenia techniczne po stronie urządzeń mobilnych uniemożliwiły skorzystanie z JMS. Jest to technologia wymagająca znacznych zasobów, niedostępnych w specyfikacji Java Micro Edition - platformy na której stworzyliśmy nasz szablon. Nie daje ona możliwości bezpośredniego podłączenia do kanału komunikacyjnego, co wymusza korzystanie z zewnętrznych bibliotek. Jednak nawet wtedy, okazuje się, że takie biblioteki korzystają z Web Serwisów jako technologii pośredniczącej w komunikacji. Podobnie, automatyczna serializacja i deserializacja w takiej sytuacji nie jest jeszcze wspierana.
Z tych powodów zdecydowaliśmy się na wykorzystanie Web serwisów jako warstwy komunikacyjnej i samodzielne rozwiązanie problemów, które się z tym wiążą.

Aplikacja serwerowa

Przedstawimy teraz architekturę części serwerowej, komponenty z których się składa i technologie, które wykorzystują.
Aby zrozumieć projekt całego systemu wprowadźmy pewne pojęcia w naszym systemie, z których wynikają dalsze rozwiązania:

- *Ankieta* - treść, dla której istnieją dostępne zdefiniowane odpowiedzi.
- *Odpowiedź* - jedna z możliwości wyboru dostępnych dla danej ankiety
- *Użytkownik* - konto z którym związane są uprawnienia i udzielone w ankietach odpowiedzi. Jeden użytkownik może udzielić wielu odpowiedzi w trakcie działania całego systemu.

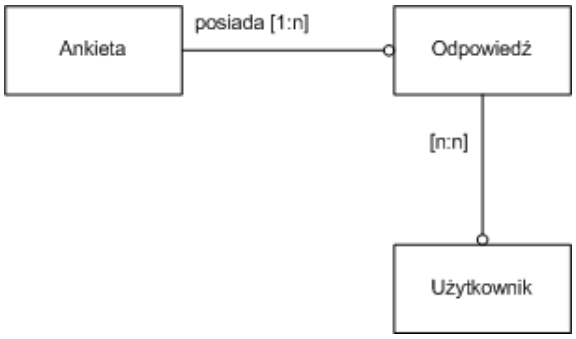
Aplikacja serwerowa zbudowana została w oparciu o model trójwarstwowy, z podwójnym dostępem (widokiem) do danych.



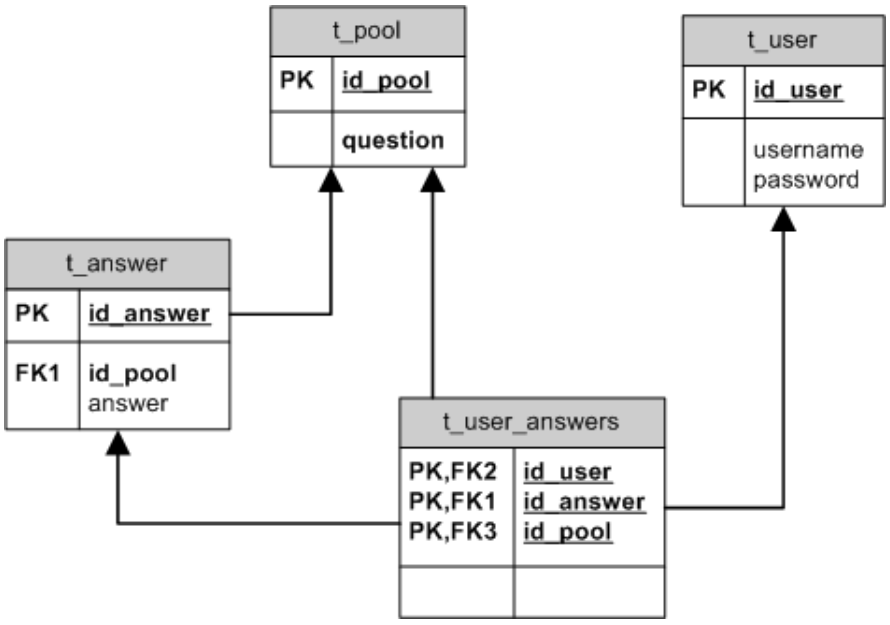
Przedstawmy teraz poszczególne warstwy systemu, wraz z przyjętymi w nich założeniami.

Serwer Aplikacja serwerowa jest uruchamiana na kontenerze Apache Tomcat 6.0.

Baza danych W oparciu o przedtawione wcześniej założenia stworzony został prosty schemat bazy danych:



Choć wydaje się, że ogranicza to działanie systemu do możliwości wyboru odpowiedzi z dostępnej listy, to jednak schemat bez problemu można rozbudować o dodatkowe pola, które pozwolą na podawanie wolnych (nieograniczonych dostępną listą wyboru) odpowiedzi.
Na postawie schematu koncepcyjnego stworzony został schemat fizyczny bazy danych:



Hibernate Dostęp do danych realizowany jest za pomocą mapowania relacyjno-obiektowego Hibernate. Teoretycznie pozwala to, a na pewno znacznie ułа-

twia ewentualną zmianę dostawcy bazy danych. Dodatkowo uwalnia nas z konieczności ręcznego pisania zapytań SQL.

Spring Framework Spring jest zestawem narzędzi ułatwiającym tworzenie aplikacji. Pozwala on, między innymi, na deklaratywne zarządzanie transakcjami, zarządzanie czasem cyklem życia obiektów oraz zwalnia programistę z konieczności pisania często powtarzanego kodu przez dostarczenie odpowiednich szablonów. Dodatkowo wymusza on pewien model programowania i sprawia, że pisane aplikacje są bardziej przejrzyste i łatwiejsze w utrzymaniu. Jak już zostało to wspomniane aplikacja serwerowa jest aplikacją typu Web, z którą to framework Spring doskonale się integruje.

Apache Myfaces Myfaces jest implementacją specyfikacji Java Server Faces, która pozwala na wygodne tworzenie widoku aplikacji. W naszym przypadku interfejsem będzie dynamicznie generowana strona WWW, choć specyfikacja JSF nie wprowadza takiego ograniczenia. Warstwa ta będzie służyła do administrowania systemem, przeglądania podjętych decyzji i zarządzania ankietami i użytkownikami.

Apache Axis2 Axis2 to kontener webserwisów,który odpowiada za odbieranie zdalnych wywołań procedur i przekazywanie ich do odpowiednich obiektów które je wykonują. Zwalnia też programistę z konieczności samodzielnego przetwarzania XML w żądaniach, umożliwia też automatyczną serializację i deserializację obiektów Javowych.

Transformata XSLT Elementem wspomagającym w systemie jest transformata XSLT wykonywana na odpowiedziach przesyłanych do urządzenia mobilnego. Jej celem jest utworzenie widoku dla urządzenia mobilnego w spo-

sób niezależny od generowanej odpowiedzi. W wyniku wywołania procedury generującej ekran, tworzony jest XML, zawierający podstawowe informacje, a następnie przetwarzany jest on w opisany sposób, dzięki czemu odpowiedź można dowolnie modyfikować bez zmiany kodu aplikacji.

Aplikacja mobilna

Cechą wyróżniającą podejście prezentowane w niniejszej pracy od typowych rozwiązań problemu integracji systemów enterprise jest sposób generowania interfejsu użytkownika. W typowym podejściu po stronie aplikacji mobilnej mamy do czynienia z raz zdefinowanym przez programistę układzie oraz wyglądzie interfejsu graficznego. Wiąże się to bezpośredno ze sposobem pisania aplikacji, w którym programista ma do dyspozycji gotowe komponenty ui, które muszą zostać odpowiednio oprogramowane i wkompilowane w końcową wersję aplikacji mobilnej. To powoduje, że jakiekolwiek zmiany w interfejsie są możliwe tylko i wyłącznie poprzez przebudowę aplikacji. A co za tym idzie także i ponowne rozprowadzenie tak zmienionego programu. Takie podejście generuje dodatkowe koszty wynikające z potrzeby zapewnienia dostarczenia aktualnej wersji aplikacji do wszystkich klientów posiadających jej starą wersję. Częściowo problem ten rozwiązuje podejście angażujące cinkiego klienta w postaci mobilnej przeglądarki internetowej. Zapewnia ono centralizację aplikacji przez co wdrożenie nowych wersji staje sie niemal automatyczne. Niestety rozwiązanie tego typu jest często niewystarczająco elastyczne oraz posiada słabo rozbudowany interfejs użytkownika. W niniejszej pracy wykorzystane zostało podejście pośrednie, będące czymś pomiędzy interfejsem generowanym przez przeglądarki internetowe, a pełnowartościowym interfejsem aplikacji mobilnych.

Dynamicznie generowany interfejs użytkownika

W odróżnieniu od typowej aplikacji dostępnej na urządzeniach mobilnych, nasza aplikacja nie posiada zdefiniowanego z góry interfejsu użytkownika. Za wyjątkiem głównego ekranu, wszystkie pozostałe są generowane na podstawie danych przesyłanych z serwera. System zachowuje się analogicznie do przeglądarki internetowej - redeneruje znaczniki interfejsu użytkowniku. W odróżnieniu od rozwiązań oprtych na przeglądarce mamy możliwość zmian w kodzie źródłowym aplikacji redenerującej, dzięki czemu w zależności od potrzeby możemy dostosowywać zachowanie programu do wymagań użytkownika. Nasza aplikacja nie jest ograniczona przez niemodyfikowalną przeglądarkę internetową.

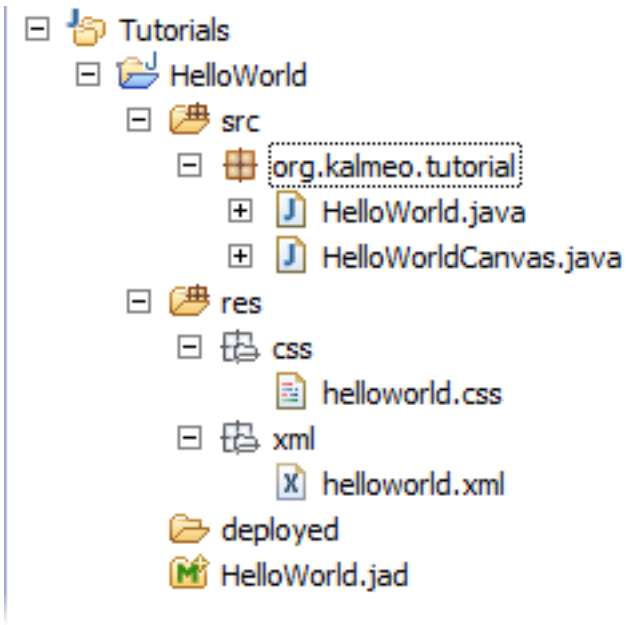
Kuix

Opisany w poprzednim rozdziale dynamicznie generowany interfejs jest możliwy do osiągnięcia w środowisku j2me przy użyciu pewnego danego z góry uniwersalnego sposobu opisu wyglądu poszczególnych elementów aplikacji. Ze względu na popularność XML oraz wsparcie dla tego języka za równo po stronie urządzeń jak i serwerów, można przyjąć, że właśnie w nim opisany zostanie interfejs użytkownika. Po stronie serwera, na podstawie informacji z bazy danych budowany będzie dokument XML łączący dane z ich sposobem wizualizacji. Po stronie urządzenia mobilnego dokument ten zostanie przetworzony przez parser XML, a następnie jego poszczególne elementy zostaną zmapowane na odpowiadające im elementy interfejsu j2me (odpowiednio rozszerzone o dodakowe komponenty, niedostępne w standardowej wersji tego środowiska). Opisana powyżej procedura została zaimplementowana w szkielecie programistycznym kuix.

Cechy szkieletu Kuix

Model programistyczny oferowany przez szkielet Kuix znacznie różni się od innych rozwiązań spotykanych na platformach mobilnych. Posiada on wiele cech charakterystycznych dla lekkiego modelu tworzenia oprogramowania mobilnego - opartego o mobilną przeglądarkę internetową. Najlepszą ilustracją zasady działania tego szkieletu, jest przykładowa aplikacja, więc w dalszej części pracy zostanie przedstawiony krótki przykładowy program obracujący podstawowe idee.

HelloKuix Na poniższym obrazku widoczną jest struktura bardzo prostgo programu posiadającego wszystkie cechy dostarczane przez szkielet Kuix.



Poza standardowymi plikami *.java zawierającymi kod aplikacji należy tu zwrócić uwagę na dodatkowe pliki zasobów :

- helloworld.css
- helloworld.xml

Plik helloworld.xml to dokument xml we wspomnianym wcześniej formacie reprezentującym dane wraz ze sposobem ich wizualizacji.

```
<screen title="helloworld">
    <container style="layout:inlinelayout(false,fill); align: center">
        <text text="Hello World!" />
        <picture src="logo_community.png" />
    </container>
</screen>
```

Tag screen to pojedynczy ekran na urządzeniu mobilnym mogący posiadać jeden, bądź więcej tak zwanych widgetów - elementów interfejsu użytkownika. Powyższy przykład zawiera kontener (container), czyli pojemnik widgetów pozwalający na grupowanie ich w odrębne układy. W pojemniku znajduje się tekst (tag text) oraz obazek (tag picture). Informacje o zawartości oraz cechach poszczególnych elementów przekazywane są za pomocą atrybutów.

Powyższy plik xml jest wczytywany na początku działania programu. Służy do tego kod widoczny poniżej.

```
// Load the content from the XML file with Kuix.loadScreen static method
Screen screen = Kuix.loadScreen("helloworld.xml", null);


// Set the application current screen
screen.setCurrent();
```

Jak widać z załączonego przykładu budowanie elementów interfejsu, oraz ich wczytywanie jest bardzo proste i pewnien sposób przypomina tworzenie interfejsu użytkownika dla stron wyświetlanych w przeglądarkach inernetowych.

CSS, a Kuix Charakterystycznym elementem zapożyczonym przez Kuix z technologii internetowych jest sposób nadawania pożądanego wyglądu elementom aplikacji. Służą do tego kaskadowe arkusze stylów (Cascading Style Sheets). W przykładzie z poprzedniego paragrafu plik helloworld.css wygląda następująco :

```
text {
    align: center;
    font-style: normal;
    color: #f19300;
}
screenTopbar text {
    color: white;
    padding: 1 2 1 2;
}
screenTopbar {
    font-style: bold;
    bg-color: #cccccc;
    border: 0 0 1 0;
    border-color: #f19300;
}
desktop {
    bg-color: #444447;
}
```

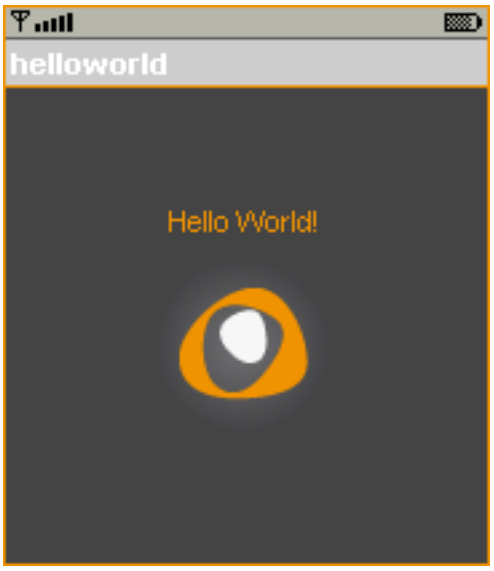
Powyższy kod css praktycznie niczym się nei różni od tego spotykanego w projektach pisanych pod standardowe przeglądarki internetowe. Do każdego elementu interfejsu, takiego jak ekran, czy pulpit (screen, desktop) mamy

przypisane odpowiednie selektory. Na przykład by nadać styl paskowi tytułowemu ekranu używamy selektora screenTopbar.

By załadować do systemu plik css znajudjący się w zasobach projektu należy użyć statycznej metody loadCSS klasy Kuix:

```
// Load the stylesheet from the CSS-like file with Kuix.loadCss static method
//  note: a stylesheet is not associated with a screen but with the midlet
//  note 2 : by default '/css/' folder is use to find the 'helloworld.css' file
Kuix.loadCss("helloworld.css");
```

Końcowy efekt połączenia pliku xml ze stylami css widoczny jest na poniższym obrazku:



W analogiczny sposób można równie łatwo dodać podręczne menu do aplikacji :

```
<screenFirstMenu>Exit</screenFirstMenu>
<screenSecondMenu>
    more...
```

```
        <menuPopup>
            <menuItem>
                About
            </menuItem>
            <menuItem>
                Exit
            </menuItem>
        </menuPopup>
    </screenSecondMenu>
```

Efekt końcowy wraz z menu dostępnym pod prawym przyciskiem (tak zwane Second Menu).

3.2.4 Istniejące problemy

Bezpieczeństwo

Gwarancja dostarczenia

Transakcyjność

Rozdział 4

Mobilny system informacyjny

Przykładem integracji systemu klasy Enterprise z platformą mobilną jest aplikacja dostarczająca spersonalizowane informacje. Ideą stojącą za stworzeniem takiego systemu jest zagospodarowanie czasu użytkowników nie mogących aktywnie wyszukiwać interesujących ich wiadomości. Z takimi sytuacjami mamy do czynienia podczas oczekiwania na środki komunikacji miejskiej, a także w czasie utrudnień w ruchu samochodowym. W takim momencie można zaproponować lekturę najnowszych informacji w skróconej formie. Dodatkowo czytelnik w prosty sposób może określić, czy aktualnie pokazywane wiadomości interesują go czy też nie. W ten sposób uzyskamy możliwość analizy zainteresowań użytkowników, tak aby w przyszłości przekazywać im jedynie informacje które mogą ich zainteresować.

4.1 Funkcjonalność systemu

Funkcjonalność systemu będzie oparta na następujących założeniach:

- Aplikacja na urządzeniu mobilnym wyświetla krótkie, dostosowane do użytkownika informacje

- Informacje zmienieniają się automatycznie, tak aby ograniczyć potrzebę samodzielnego ich wyszukiwania
- Istnieje możliwość określenia czy aktualnie pokazywana wiadomość jest dla nas interesująca, co będzie miało wpływ na dobór kolejnych wiadomości
- Możliwość robienia zakładek na wybranych wiadomościach, co spowoduje wysłanie odnośników do nich na skrzynkę pocztową, w celu późniejszego przeczytania

4.2 Wymagania systemu

Infrastruktura Potrzebne będzie połączenie z Internetem, dostępne z urządzenia mobilnego jak i z serwera dostarczającego informacje. Dodatkowo dla komfortowego przeglądania wiadomości, urządzenie mobilne powinno posiadać ekran o odpowiednio dużej rozdzielczości (co najmniej 320x240).

Oprogramowanie Urządzenie mobilne będzie musiało posiadać maszynę wirtualną Java. Aplikacja na serwerze będzie działała pod kontrolą kontenera Apache Tomcat.

4.3 Wstępna analiza projektu

Przypuszczamy, że system który planujemy stworzyć może znaleźć realne zastosowanie i odnieść sukces rynkowy.

	Zalety	Wady
Wewnętrzne	Wykorzystanie popularnej platformy Niski koszt rozwiązania	Powolna lub przerywana transmisja Zbyt mały ekran i mała ergonomia użytkowania
Zewnętrzne	Długi czas spędzany w korkach lub w środkach komunikacji w dużych miastach Podatność ludzi krótkie, podane w ciekawej formie informacje	Obawa ludzi przed korzystaniem z internetu mobilnego, jako ciągle zbyt drogiego

4.3.1 Podobne systemy

Inspiracją, która stoi za naszym pomysłem jest system wyświetlania wiadomości na stacjach i w wagonach metra. Tym co chcielibyśmy w nim udoskonalić to możliwość dostarczenia spersonalizowanych, dostosowanych do każdego czytelnika informacji.

Rozdział 5

Interfejs użytkownika mobilnego klienta

Mobilny klient naszej aplikacji jest midletem zbudowanym w oparciu o szkielet Kuix. Wersja instalacyjna składa się z dwóch plików :

- Archiwum jar zawierającego midlet
- Desktyptora jad zawierającego dane niezbędne do instalacji aplikacji w trybie OTA (Over-The-Air)

Menu głóne Po instalacji aplikacji w zakładce zawierającej listę zainstalowanych na telefonie midletów powinna pojawić się nowa aplikacji o nazwie mINFO. Po jej uruchomieniu zostaniemy przeniesieni do głównego ekranu zawierającego trzy opcje :

- Show news - rozpoczyna ładowanie danych ze wskazanego serwera
- Settings - pozwala na skonfigurowanie źródła danych (serwera)
- Exit - kończy działanie midletu

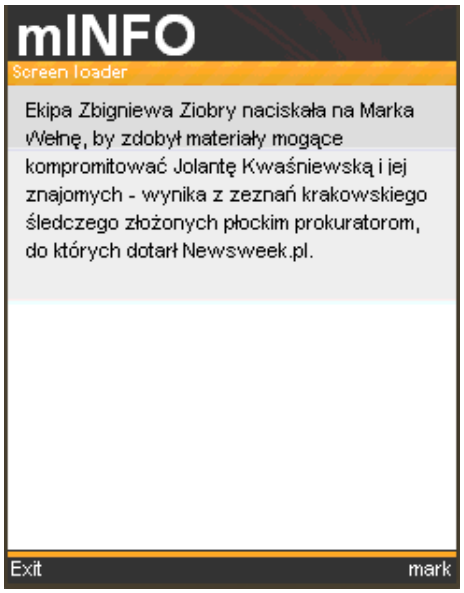
Na poniższym obrazku widzimy główne menu aplikacji. Została ona uruchomiona w domyślnym środowisku symulacyjnym dostarczonym razem z pakietem Netbeans Mobility Pack.

Zakładka Settings W zakładce settings mamy możliwość wybrania adres, pod którym znajduje się serwer dostarczający dane dla naszej aplikacji. Aplikacja potrafi zapamiętać kilka takich adresów. By dodać nowy adres należy w polu 'New feed address' wpisać URL do nowego serwera, a następnie

wybrać przycisk 'New feed'. Jeśli chcemy pozbyć się niepotrzebnych wpisów znajdujących się na liście, to zaznaczamy wpis do usunięcia, a następnie wybieramy przycisk 'Remove feed'. Istnieje także możliwość modyfikacji adresu URL serwera. Służy do tego przycisk 'Edit feed'.

Zakładka 'Settings' po wprowadzeniu nowego adresu serwera:

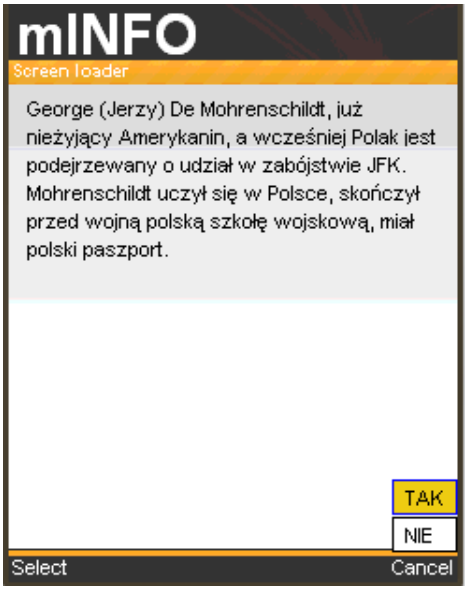
Dynamicznie generowane ekrany wiadomości Po wybraniu z głównego menu opcji 'Show news' zostajemy zalogowani jako nowy użytkownik - pod warunkiem, że pierwszy raz podłączamy się do danego serwera. W przeciwnym wypadku z pamięci urządzenia zostaje odczytana uprzednio wygenerowana nazwa użytkownika oraz hasło. Dzięki temu serwer jest w stanie śledzić działania wykonywane na danym urządzeniu przez użytkownika, poddawać je analizie, a następnie zmieniać zawartość wysyłanych treści. Ekrany z wiadomościami są całkowicie generowane po stronie serwera (nie tylko sama ich treść ale także elementy interfejsu użytkownika). Przykładowy ekran informacyjny :



Ekrany są automatycznie przeładowywane co pewien określony z góry przedział czasu. Dzięki temu jeśli użytkownik chce, to może biernie wykorzystywać naszą aplikację (w trybie nie wymagającym interakcji).

Wybór preferencji odnośnie treści wiadomości Jeśli użytkownik chce wykorzystywać aplikacje w sposób interaktywny, to ma do dyspozycji pod-

ręczne menu określenia preferencji dotyczących aktualnie oglądanej wiadomości. Może poinformować system, że dana wiadomość go szczególnie zainteresowała, lub w przeciwnym wypadku, że nie chce więcej dostawać tego typu wiadomości. Na podstawie udzielonych odpowiedzi system personalizuje listę wiadomości, które zostaną wysłane do danego użytkownika.



Bibliografia

[1] Michael Juntao Yuan: *Enterprise J2ME Developing Mobile Applcations*, Prentice Hall Pennsylvania 2004

[2] Gregor Hophe,Bobby Woolf: *Enterprise Integration Patterns*, The Addison-Wesley Signaure Series 2003

[3] : *Spring Tutorial*, http://www.techfaq360.com/tutorial/spring/

[4] : *Which WSDL*, http://www.ibm.com/developerworks/webservices/library/ws-whichwsdl/