Celem zadań będzie napisanie aplikacji agregującej wpisy z różnych stron newsowych (udostępniających wpisy poprzez RSS). Aplikacja będzie budowana stopniowo podczas szkolenia, wraz z kolejnymi aspektami platformy JEE6 przedstawianymi podczas szkolenia.

Agregacja artykułów będzie wykorzystywać ideę *pubsubhubbub (PuSH)*, czyli protokół umożliwiający bardzo szybkie dostarczenie wiadomości do czytnika RSS.

A simple, open, server-to-server web-hook-based pubsub (publish/subscribe) protocol as an extension to Atom and RSS.

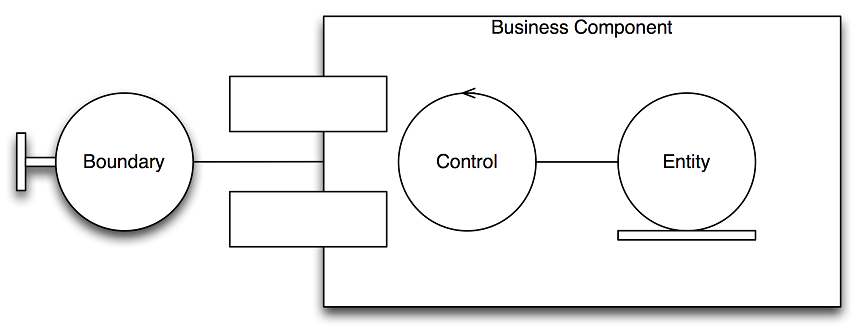
Parties (servers) speaking the [PubSubHubbub protocol](http://pubsubhubbub.googlecode.com/svn/trunk/pubsubhubbub-core-0.2.html) can get near-instant notifications (via webhook callbacks) when a topic (feed URL) they're interested in is updated.

Nasza aplikację będzie miała za zadanie agregować tak dostarczane wpisy: odbierać je, przetwarzać, zapisywać w bazie danych, udostępniać podstawowe statystyki poprzez kilka różnych formatów (lub nawet sposobów) – przez wszystkie te etapy przejdziemy podczas szkolenia.

# Ćwiczenie 1

Na początek stworzymy kilka komponentów zajmujących się przetwarzaniem (a dokładnie parsowaniem) samej wiadomości.

Zacznijmy od grama teorii, w ogólności niemal każdą funkcjonalność implementuje „wzorzec” Entity, Control, Boundary i można ją przedstawić na prostym schemacie blokowych:



Odpowiedzialność każdego elementu jest dość ściśle określona:

* **Boundary**: fasada, element będący wejściem do systemu dla interfejsu użytkownika (bądź jakiejkolwiek innej interakcji z użytkownikiem)
* **Control**: to jest serwis (lub zestaw serwisów) oddzielonych od zewnętrznego świata; mogą być to skąp likowane serwisy bądź proste, generyczne operacje typu CRUD
* **Entity**: obiekty związane z domeną; w przypadku ogólnym encję JPA, ale może to być wyższy poziom abstrakcji (DAO) lub niższy (bezpośrednie operacje na bazie danych) – wszystkie w zależności od przypadku użycia.

Opierając się na powyższym schemacie możemy przystąpić do implementacji naszej aplikacji. W pierwszy kroku zaprojektujemy i zaimplementujemy fragment, który poprzez prosty interfejs użytkownika (pole typu text area), przekaże do serwisu artykuł w formie RSS. Artykuł ten zostanie następnie przetworzony i w konsoli zostaną wyświetlone podstawowe informacje zawarte w nagłówku.

Odczyt RSSu może odbywać się bibliotekę Rome

<dependency>

<groupId>rome</groupId>

<artifactId>rome</artifactId>

<version>0.9</version>

</dependency>

String text = //...

SyndFeedInput input = **new** SyndFeedInput();

SyndFeed feed = input.build(**new** StringReader(text));

# Ćwiczenie 2

Skoro nasza aplikacja potrafi przetworzyć podstawowy artykuł przejdźmy kolejnego kroku; biblioteka Rome jest dość ułomna. Ze względu na swój wiek nie wykorzystuje takich funkcjonalności jak *Generics* przez co korzystanie z niej jest utrudnione. Aby zminimalizować ilość potrzebnego rzutowania dodajmy do naszego pakietu konwerter, który zmapuje obiekty biblioteki Rome na model wykorzystywany w aplikacji.



# Ćwiczenie 3

Tak przygotowany model możemy wyświetlić w postaci prostej listy:

* [Nazwa kanału]

1. [Nazwa elementu 1]
2. [Nazwa elementu 2]
3. …..

W najprostszym przypadku, nasz serwlet połączony jest z tym samym komponentem EJB, więc zwykłe przechowanie listy jako prywatnego pola w komponencie będzie wystarczające do realizacji tego zadania. Zostawmy to na taki etapie, pomimo że przechowywanie stanu (co właśnie zrobiliśmy) w komponencie bezstanowym i korzystanie z niego pomiędzy wywołaniami jest niebezpiecznie i nie powinno być pod żadnym pozorem stosowane w aplikacjach produkcyjnych.

# Ćwiczenie 4

W kolejnym kroku przygotujemy obsługę metody callback (tzw. webhook) który zostanie wywołany aby przekazać do naszej aplikacji informację o nowym artykule (wraz z samym artykułem). W tym celu, posługując się analogicznym wzorcem jak w ćwiczeniu pierwszym – przygotowujemy endpoint zgodny z JAX-RS o następującej strukturze:

http://localhost:8080/aplikacja/feeds/{identyfikator kanału}

Na powyższy URL, metodą POST wysyłane będą dane, gdzie cały „feed” zawarty będzie w ciele zapytania (HttpMessageBody).

Taki przesłany feed przetwarzamy; wynik operacji zobaczymy w konsoli serwera.

# Ćwiczenie 5

Aby zobaczyć wyniki nie poprzez konsolę (ale np. na ekranie przeglądarki) musimy w spójny sposób zacząć przechowywanie stanu aplikacji. Na obecną chwilę dwa różne komponenty EJB są połączone z dwoma różnymi wejściami do aplikacji: serwlet i usługa REST (możemy to doskonale zaobserwować logując wywołania metody @PostConstruct komponentu EJB).

Najprostszym sposobem aby synchronizować dane pomiędzy tymi komponentami jest utworzenie kolejnego komponentu – lokalnego cache’a. W tym celu utworzymy nowy komponent będący singletonem i tam będziemy przechowywać dane.

# Ćwiczenie 6

W kolejnym kroku możemy pokusić się o zapisywanie danych do bazy. W tym celu należy skonfigurować połączenie do bazy danych oraz odpowiednio zmapować encję. Wtedy będzie możliwe ich zapisanie (przy pomocy EntityManagera).

Zamiast w lokalnej pamięci serwera (poprzedni komponent będący singletonem) zapiszmy dane do bazy.

Skorzystajmy z lokalnej bazy MySQL, które jest już prekonfigurowana na serwerze aplikacji (nazywa się MySQL DataSource).

# Ćwiczenie 6a

Aby zapewnić poprawność danych, należy je przed zapisem sprawdzić (oraz nałożyć odpowiednie więzy poprawności zgodnie ze standardem BeanValidation).

W tym kroku dodaj sprawdzanie poprawności zapisywanych danych (po stronie zarówno serwletu, jak komponentu webowego). Zgodnie z wytycznymi komunikacji REST, w przypadku niepowodzenia zapisu danych należy zwrócić odpowiedni nagłówek http: 422 – pozostający zgodny z opisem RFC

The 422 (**Unprocessable Entity**) status code means the server understands the content type of the request entity (hence a 415(Unsupported Media Type) status code is inappropriate), and the syntax of the request entity is correct (thus a 400 (Bad Request) status code is inappropriate) but was unable to process the contained instructions. For example, this error condition may occur if an XML request body contains well-formed (i.e., syntactically correct), but semantically erroneous, XML instructions.

# Ćwiczenie 7

Jednym z wymagań jest udostępnienie serwisom zewnętrznym statystyk dotyczących dat oraz ilości przetworzonych aplikację wpisów. Dane te należy udostępnić poprzez usługę sieciową typu SOAP.

Klient wywołując naszą usługę podaje swój identyfikator, który jest zapisywany w systemie w sposób trwały, ponieważ posłuży do późniejszego rozliczenia się z klientem. Jako że jest to usługa komercyjna, logujemy także dane, które zwracane są do klienta.

Dane zwracane są w formie zagregowanej: data, feed, ilość wywołań w dniu, ilość artykułów pobranych w dniu.

Ponieważ ta funkcjonalność jest dość krytyczna dla naszych klientów, wszelkie logowania muszą odbywać się niezależnie i w tle głównego przetwarzania (mają niższy priorytet).

Hint: jako że wymaganie logowanie może się zmieniać nie powinno być trwale związane z naszą logiką biznesową.