# Rozdział

# Architektura nowoczesnych aplikacji internetowych

Lech Madeyski, Michał Stochmiałek
*Wydziałowy Zakład Informatyki, Wydział Informatyki i Zarządzania*
*Politechnika Wrocławska*
*Lech.Madeyski@pwr.wroc.pl, Michal.Stochmialek@e-informatyka.pl*

*Any technology distinguishable from magic is insufficiently advanced.*
Gregory Benford

**Streszczenie**

*Projekt architektoniczny dotyczy decyzji, mających na celu uzyskanie systemu o odpowiedniej charakterystyce, która może mieć wiele znaczeń np. utrzymywalność (ang. maintainablility), czy skalowalność (ang. scalability). Istniejące szkielety architektoniczne, takie jak MVC czy PCMEF, umożliwiają tworzenie dobrze ustrukturyzowanych aplikacji, a w konsekwencji minimalizacje zależności międzymodułowych. W niniejszym rozdziale podjęto próbę analizy tych wzorców w kontekście ich zastosowania w aplikacjach internetowych. Rezultatem prac jest propozycja szkieletu architektonicznego XWA (ang. eXtensible Web Architecture), prezentacja jego praktycznej implementacji na przykładzie serwisu e-Informatyka oraz zalecenia dotyczące tworzenia aplikacji internetowych na bazie szkieletu publikacji Apache Cocoon, jak i innych rozwiązań zbliżonych technologicznie.*

## 1. Wprowadzenie

Wybór odpowiedniej architektury dla tworzonego systemu stanowi poważne wyzwanie szczególnie w przypadku dużych aplikacji internetowych wykorzystujących najnowsze możliwości technologiczne. Ma bowiem istotny wpływ m.in. na łatwość pielęgnacji,

czyli utrzymywalność (ang. *maintainablility*) systemu czy jego skalowalność (ang. *scalability*).

W rozdziale tym przedstawiono dwa istniejące wzorce czy szkielety architektoniczne[1] (ang. *architectural framework*): MVC i PCMEF, bazującą na nich własną propozycję szkieletu architektonicznego XWA (ang. *eXtensible Web Architecture*) uwzględniającego specyfikę aplikacji internetowych, jego praktyczną implementację na przykładzie serwisu e-Informatyka oraz zalecenia dotyczące tworzenia aplikacji internetowych na bazie szkieletu publikacji Apache Cocoon, jak i jak i innych rozwiązań zbliżonych technologicznie.

Sekcja 2. zawiera szczegółowe omówienie zarówno klasycznego wzorca MVC, jak również niewątpliwie interesującego, bazującego na strukturze warstwowej, szkieletu PCMEF.

W sekcji 3. skonfrontowano powyższe szkielety architektoniczne ze specyfiką aplikacji internetowych.

Sekcja 4. jest próbą uwzględnienia powyższej specyfiki i stanowi szczegółowe omówienie własnej propozycji szkieletu architektonicznego XWA, jak również jego praktycznej implementacji.

Sekcja 5. zawiera podsumowanie rezultatów prac.

# 2. Wybrane podejścia do kwestii architektury

Poniżej przedstawiono szkielety architektoniczne MVC oraz PCMEF. Te dwa podejścia wywarły istotny wpływ na przedstawioną w sekcji 4. propozycję architektury aplikacji internetowych. Jest ona bowiem efektem ewolucji wspomnianych podejść.

## 2.1. Triada MVC

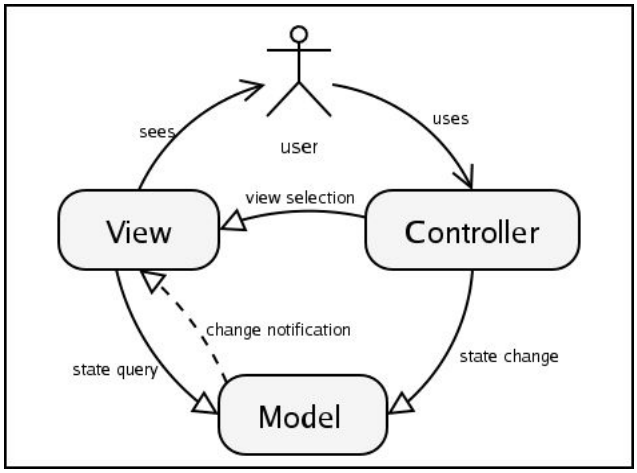
Źródła MVC (*Model-View-Controller*) sięgają końca lat siedemdziesiątych i języka Smalltalk-80. Paradygmat MVC stał się centralną koncepcją realizacji interfejsów użytkownika w tym języku, jak również w innych językach i systemach zorientowanych obiektowo [Burb1987, Fowl2003]. Jest również znakomitą ilustracją zasady podziału odpowiedzialności (ang. *separation of concerns*). Zasada ta znajduje odzwierciedlenie w podziale klas aplikacji na trzy grupy: model (ang. *model*), widok (ang. *view*) i kontroler (ang. *controller*), co obrazuje rysunek 1[2].

[1] Zdaniem autorów pojęcie szkieletu architektonicznego [Mac2004] jest w dużym stopniu zbieżne z pojęciem wzorca architektonicznego [Busc1996].

[2] Z powodu problematycznego tłumaczenia na język polski zaproponowanych przez prof. Maciaszka nazw pakietów szkieletu PCMEF oraz w celu zachowania spójnosci językowej pomiędzy rysunkami niniejszego rozdziału autorzy zachowali nazewnictwo angielskie.

Semantyka triady MVC i reguły komunikacji pomiędzy jej poszczególnymi elementami są następujące.

**Model** związany jest z domeną aplikacji, zawiera jej elementy statyczne i behawioralne. Najważniejszą składową modelu jest logika aplikacji oraz stojąca za nią logika biznesowa. Innymi słowy, model określa działanie aplikacji oraz przetwarzane dane. Powinien być tak zaprojektowany, żeby był niezależny od wybranego rodzaju prezentacji oraz systemu obsługi akcji użytkownika (*model nic nie wie o widoku i kontrolerze*). Jedyne powiązanie wychodzące z modelu to powiadomienie widoku o aktualnych zmianach (*change notification*). W większości implementacji jest to rozwiązane za pomocą systemu komunikatów lub z zastosowaniem wzorca *Observer* [Gamm1996].




Rys. 1. Szkielet architektoniczny MVC (klasyczny)

**Widok** zarządza graficzną lub tekstową prezentacją modelu (jest jego wizualnym odwzorowaniem). Realizacja widoku jest już powiązana z konkretnym modelem. Prezentacja nie może zostać wygenerowana bez znajomości specyfiki danych czy operacji, które obrazuje użytkownikowi. Rysunek 1. ilustruje to powiązanie: widok pobiera informacje z modelu (*state query*) ilekroć zostaje powiadomiony o jego zmianie. Z drugiej strony model nie jest związany ze sposobem jego prezentacji. W związku z tym zmiany widoku nie pociągają za sobą zmian w modelu.

**Kontroler** jest natomiast odpowiedzialny za reagowanie na akcje użytkownika (np. kliknięcia myszką), odwzorowując je na operacje zawarte w modelu (*state change*) oraz na zmiany widoku (*view selection*). W połączeniu z widokiem odpowiada za dwa aspekty interfejsu użytkownika (*look and feel*).

Praktyka niestety pokazuje, że ten element triady jest często różnie interpretowany [Fowl2003]. Bywa mylnie uważany za element reprezentujący aspekt behawioralny (działanie aplikacji) przy jednoczesnym założeniu, że model reprezentuje aspekt statyczny (dane). Tak rozumiany kontroler, jednocześnie odpowiedzialny za sterowanie widokiem (logika przetwarzania żądań użytkownika), nie pozwala na zmianę warstwy prezentacji bez modyfikacji logiki aplikacji (znajdującej się w tym przypadku

w kontrolerze). Z tego względu kontroler nie powinien zawierać logiki aplikacji, jedynie odwołania do niej.

Wartość szkieletu architektonicznego MVC leży w dwóch podstawowych zasadach. Pierwsza to **separacja prezentacji i modelu**, umożliwiająca zmianę interfejsu użytkownika (np. udostępnienie usług aplikacji poprzez interfejs graficzny i tekstowy). Druga zasada to **separacja widoku i kontrolera**. Klasycznym przykładem jest użycie dwóch kontrolerów związanych z jednym widokiem, które umożliwiają edycję widoku lub jej nie umożliwiają [Fowl2003].

Klasyczna postać triady MVC jest jednak często zdegenerowana. Degeneracja ta polega na mocnym połączeniu kontrolera i widoku. Znalazło to swoje odzwierciedlenie m.in. w niektórych wersjach języka Smalltalk, jak również w bibliotece Swing na platformie Java [Fowl1998]. Projektanci biblioteki Swing zrezygnowali bowiem z dokładnego stosowania szkieletu MVC na rzecz architektury Model-Delegat (ang. *Model-Delegate*).

Specyfika aplikacji internetowych sprawia jednak, że powraca się do klasycznej postaci szkieletu architektonicznego MVC. Zostanie to szczegółowo opisane w sekcji 3.1.

## 2.2. Szkielet architektoniczny PCMEF

W projektach architektonicznych często używa się struktur warstwowych. Dobrze zaprojektowane pozwalają na pogrupowanie klas w pionową hierarchię oraz na minimalizację zależności pomiędzy pakietami. Stabilność architektury polega na tym, że zmiany w warstwach wyższych nie wymuszają modyfikacji warstw niższych. Tak osiągnięta stabilność jest możliwa pod warunkiem, że dozwolone są tylko zależności warstw wyższych od niższych (tylko zależności *w dół*). Z powodów praktycznych często jednak dopuszcza się luźne zależności (ang. *loose coupling*) warstw niższych od wyższych (bazujące na komunikatach lub interfejsach).

PCMEF, określany mianem warstwowego szkieletu architektonicznego, składa się z czterech warstw: `presentation`, `control`, `domain` i `foundation` [Mac2003a, Mac2003b, Maci2004]. Warstwa `domain` składa się z dwóch pakietów: `entity` i `mediator`. Ich rozmieszczenie przedstawia rysunek 2.

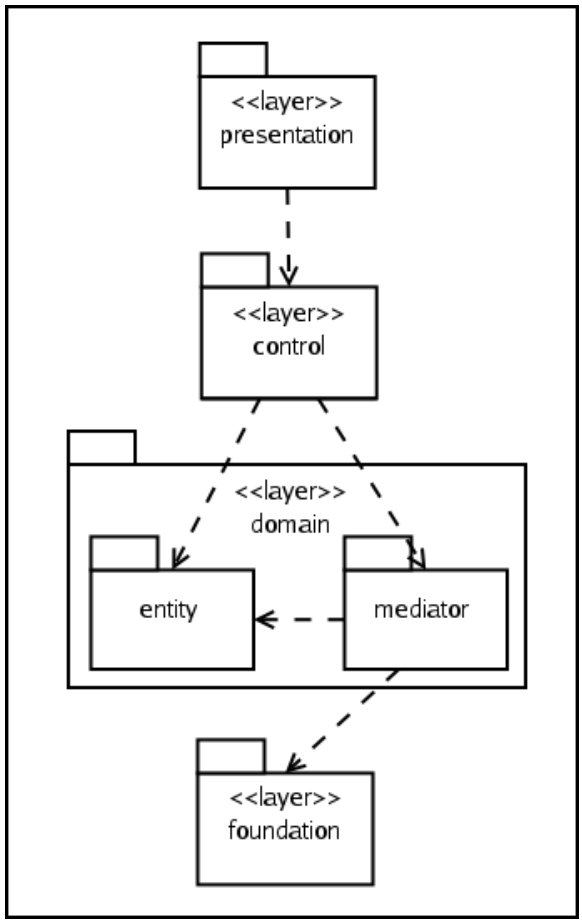
Warstwa `presentation` odpowiada za sposób prezentacji aplikacji. Składa się głównie z klas rozszerzających komponenty graficznych interfejsów użytkownika. Na przykład programując w języku Java, będą to klasy rozszerzające komponenty biblioteki Swing lub SWT. Elementem triady MVC, odpowiadającym warstwie `presentation`, jest widok silnie połączony z kontrolerem. Z podobną sytuacją mamy do czynienia w przypadku wzorca Model-Delegat w bibliotece Swing.

Warstwa `control` odpowiada za obsługę żądań użytkownika przekazanych z wyższej warstwy. Zawiera głównie klasy odpowiedzialne za logikę programu. Przykładem klas należących do tej warstwy są implementacje interfejsów słuchaczy (ang. *listeners*), będących składowymi biblioteki Swing.

Pakiet `entity`, należący do warstwy `domain`, składa się z klas reprezentujących obiekty biznesowe. Są to obiekty, które zawierają operacje i dane biznesowe. Część z nich jest trwała (ang. *persistent objects*) i posiada odwzorowanie w zewnętrznych źródłach danych (np. w bazie danych).

Pakiet `mediator`, należący do warstwy `domain`, pośredniczy pomiędzy pakietami `control` i `entity` a pakietem `foundation`. Jego wprowadzenie ma na celu usunięcie zależności `entity` od `foundation`. Zlikwidowano w ten sposób konieczność modyfikacji obiektów biznesowych przy zmianie mechanizmów trwałości danych [Maci2004]. Umożliwiono również oddzielenie konstrukcji zapytań do zewnętrznych źródeł danych od logiki aplikacji zawartej w warstwie `control`.

Warstwa `foundation` zawiera klasy odpowiedzialne za obsługę zewnętrznych źródeł danych, takich jak bazy danych, repozytoria dokumentów, usługi sieciowe (ang. *web services*) czy systemy plików.




Rys. 2. Szkielet architektoniczny PCMEF

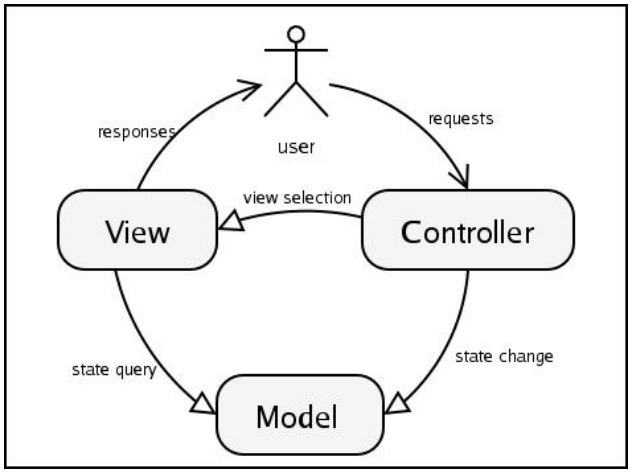
# 3. Architektura aplikacji internetowych

Każdą nową koncepcję czeka po narodzeniu najtrudniejsze wyzwanie. Musi zmierzyć się z rzeczywistością, w jakiej ma zaistnieć, np. uwzględniając jej specyfikę, której zarys, w przypadku projektowania aplikacji internetowych, przedstawiono poniżej:

- Protokół HTTP, na którym bazują aplikacje internetowe jest bezstanowy. Każdorazowa obsługa żądania użytkownika (ang. *request*), z punktu widzenia protokołu, jest niezależna i niepowiązana z poprzednimi żądaniami. To zadaniem aplikacji jest przenoszenie jej stanu pomiędzy żądaniami, jeżeli zachodzi taka potrzeba.
- Konstrukcja protokołu HTTP sprawia, że konwersacja pomiędzy użytkownikiem a systemem (serwerem) może być inicjowana tylko przez użytkownika.
- W aplikacjach internetowych przepływ sterowania (ang. *control flow*) jest kontrolowany przez żądania użytkownika i obejmuje, często rozbudowane, sekwencje interakcji użytkownika z systemem.

## 3.1. Szkielet architektoniczny MVC a aplikacje internetowe

Aktualizacja widoku jest sterowana żądaniami użytkownika. Wymusza to modyfikację klasycznej postaci szkieletu MVC.




Rys. 3. Szkielet architektoniczny MVC (aplikacje internetowe)

Z powodu braku równoprawnej konwersacji w aplikacjach internetowych nie ma sposobu na powiadomienie widoku o zmianach modelu, a wymagane aktualizacje muszą zostać przesunięte w czasie do obsługi kolejnego żądania użytkownika[3]. Rozwiązanie to, nazywane trybem pasywnym, przedstawia rysunek 3.

[3] Istnieje techniczna możliwość ominięcia tego ograniczenia. Widok może wymusić okresowe odpytanie kontrolera czy zaszły zmiany w modelu.

## 3.2. Szkielet architektoniczny PCMEF a aplikacje internetowe

Rozważane wcześniej aspekty architektury aplikacji internetowych były związane z protokołem HTTP. Należy jednak pamiętać, że często pod otoczką prezentacji, zazwyczaj w postaci stron HTML, kryje się złożona logika biznesowa, obsługa zewnętrznych źródeł danych, czy usług sieciowych. Dlatego aplikacja, aby spełniać wymagania związane z utrzymywalnością (ang. *maintainablility*) czy skalowalnością (ang. *scalability*) musi posiadać dobrze zdefiniowaną, wewnętrzną strukturę.

Niestety omówiony wcześniej wzorzec *Model-View-Controller* nie dostarcza żadnych wskazówek w tym zakresie. Prowadzi to do powstawania trudnych do opanowania sieciowych struktur zależności pomiędzy obiektami z różnych pakietów, które rosną w sposób wykładniczy podczas rozwoju systemu [Maci2004].

Sprawdzonym rozwiązaniem tego problemu jest wprowadzenie struktury hierarchicznej. Jest to zalecenie przyjęte między innymi w systemach korporacyjnych bazujących na platformie J2EE [Alur2003]. Dobrym przykładem architektury hierarchicznej jest również omówiony wcześniej szkielet architektoniczny PCMEF. Posiada on dobrze zdefiniowaną semantykę poszczególnych warstw, będąc jednocześnie na tyle ogólnym, że znajduje zastosowanie w technologicznie różnych systemach.

Pojawia się pytanie, w jakim stopniu szkielet PCMEF wykorzystuje podstawowe zasady MVC: separację prezentacji i modelu oraz separację widoku i kontrolera. Pierwsza zasada znalazła swoje wyraźne odzwierciedlenie w wyodrębnieniu warstwy prezentacji. Z kolei druga zasada nie została wyrażona w szkielecie architektonicznym PCMEF. Tak jak w bibliotece Swing, widok i kontroler są ze sobą mocno powiązane. W przypadku klasycznych aplikacji zasada ta jest często pomijana ze względów praktycznych. Jednak w przypadku aplikacji internetowych oddzielenie kontrolera od widoku jest dobrą praktyką, która powinna znaleźć swoje odzwierciedlenie w architekturze.

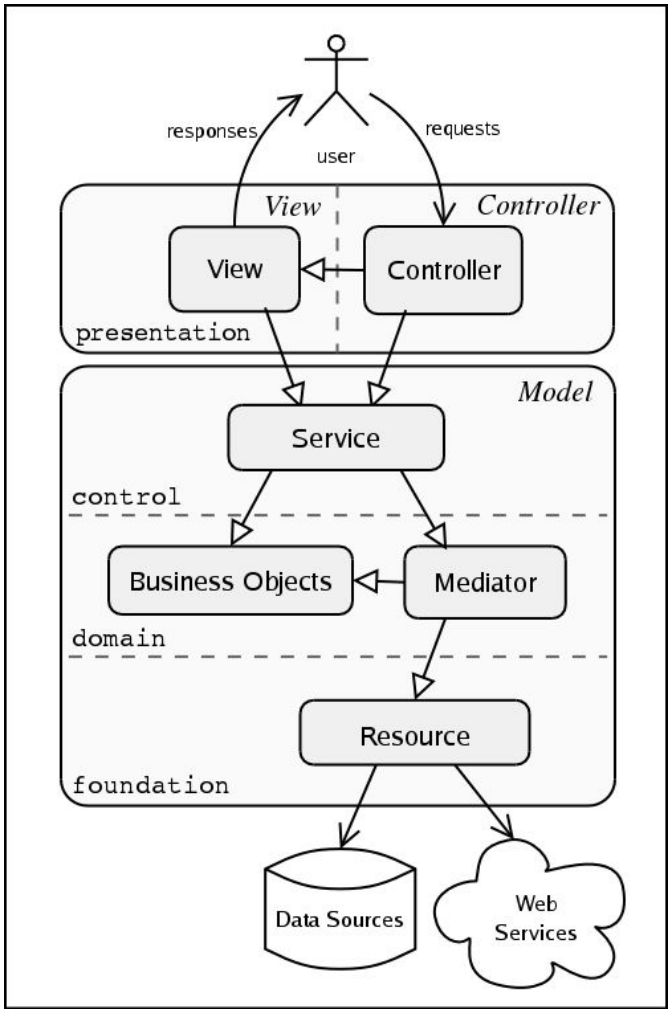
# 4. Propozycja szkieletu architektonicznego XWA

Autorzy zaproponowali szkielet architektoniczny XWA (*eXtensible Web Architecture*), który jest wariacją omówionych wcześniej rozwiązań, dostosowany do specyfiki aplikacji internetowych. Łączy ich zalety zakładając separację klas widoku od kontrolera (zgodnie z MVC) oraz podział klas wewnętrznych modelu na strukturę hierarchiczną (zgodnie z PCMEF), wykorzystując efekt synergii, co ilustruje rysunek 4.

XWA jest architekturą warstwową z wyraźnie wyodrębnioną triadą MVC. Składa się z sześciu pakietów ułożonych w czteropoziomową hierarchię ilustrującą zależności między pakietami (warstwy wyższe zależą od niższych). Zależności odwrotne są zminimalizowane do luźnych powiązań. Poniżej przestawiono opisy poszczególnych pakietów.

Pakiet **View** odpowiada za prezentację aplikacji. Jest to dokładny odpowiednik widoku w szkielecie MVC. W aplikacjach internetowych składa się z plików opisujących wygląd

stron WWW. W zależności od technologii mogą to być szablony w postaci plików HTML czy strony JSP w architekturze Model 2 [Sesh1999]. Jednak najciekawszym rozwiązaniem wydaje się, użycie technologii bazujących na metajęzyku XML w połączeniu z transformatami XSL. W takim przypadku pojawia się nowy (bazujący na XML) rodzaj kontraktów pomiędzy warstwami. Na przykład kontrakt pomiędzy widokiem a logiką aplikacji określa schemat dokumentu XML (wyrażony za pomocą DTD, XML Schema czy Relax NG).




Rys. 4. Szkielet architektoniczny XWA

Pakiet **Controller** jest odpowiedzialny za obsługę akcji użytkownika poprzez wywołanie logiki zawartej w niższych warstwach. Jego zadaniem jest oddzielenie specyfiki protokołu HTTP od logiki aplikacji. Jest on odpowiedzialny za sterowanie przepływem w ramach pojedynczej interakcji, a w przypadku zaawansowanych aplikacji internetowych odpowiada za sterowanie sekwencją interakcji. Jest to realizowane za

pomocą kontrolera kontynuacji omówionego w kolejnej sekcji. Istotnym zadaniem pakietu **Controller** jest również sterowanie widokiem. W pakiecie tym mogą znaleźć zastosowanie takie wzorce, jak *Front Controller* czy *Application Controller* [Alur2003].

Pakiet **Service** jest odpowiedzialny za udostępnienie usług systemu. Centralizuje logikę aplikacji bazującej na wielu obiektach biznesowych oraz wymagającej dostępu do zewnętrznych źródeł danych czy usług internetowych (np. poczta elektroniczna). Klasy tego pakietu mogą realizować wzorce projektowe *Application Service* [Alur2003] czy *Service Layer* [Fowl2003].

Pakiet **Business Objects** zawiera obiekty biznesowe, które tworzą model domenowy aplikacji. Ich implementacja może bazować na wzorcach *Business Object* [Alur2003] lub *Domain Model* [Fowl2003].

Pakiet **Mediator** jest warstwą pośredniczącą w dostępie do zewnętrznych źródeł danych, mechanizmów trwałości danych czy usług sieciowych. Klasy tego pakietu zazwyczaj realizują wzorzec *Data Access Object* [Alur2003].

Pakiet **Resource** realizuje niskopoziomową obsługę dostępu do zewnętrznych zasobów.

## 4.1. Architektura e-Informatyki jako implementacja XWA

Pomysł serwisu e-Informatyka (www.e-informatyka.pl) pojawił się w czerwcu 2002 roku. Postawiono sobie za cel realizację oraz wypromowanie czasopisma naukowego posiadającego mocne wsparcie w postaci elektronicznego systemu publikacji. Serwis ma stać się również składnicą wiedzy i skupiać społeczność informatyczną.

Akademicki charakter projektu, sposób jego realizacji oraz opracowane w trakcie projektu nowe podejście do specyfikacji wymagań osadzone na gruncie metodyki XP (ang. *eXtreme Programming*) zostało opisane w [Made2003a]. W chwili obecnej z technicznego punktu widzenia projekt jest w znacznym stopniu zaawansowany (m.in. jest już dostępny zwiastun nowego czasopisma).

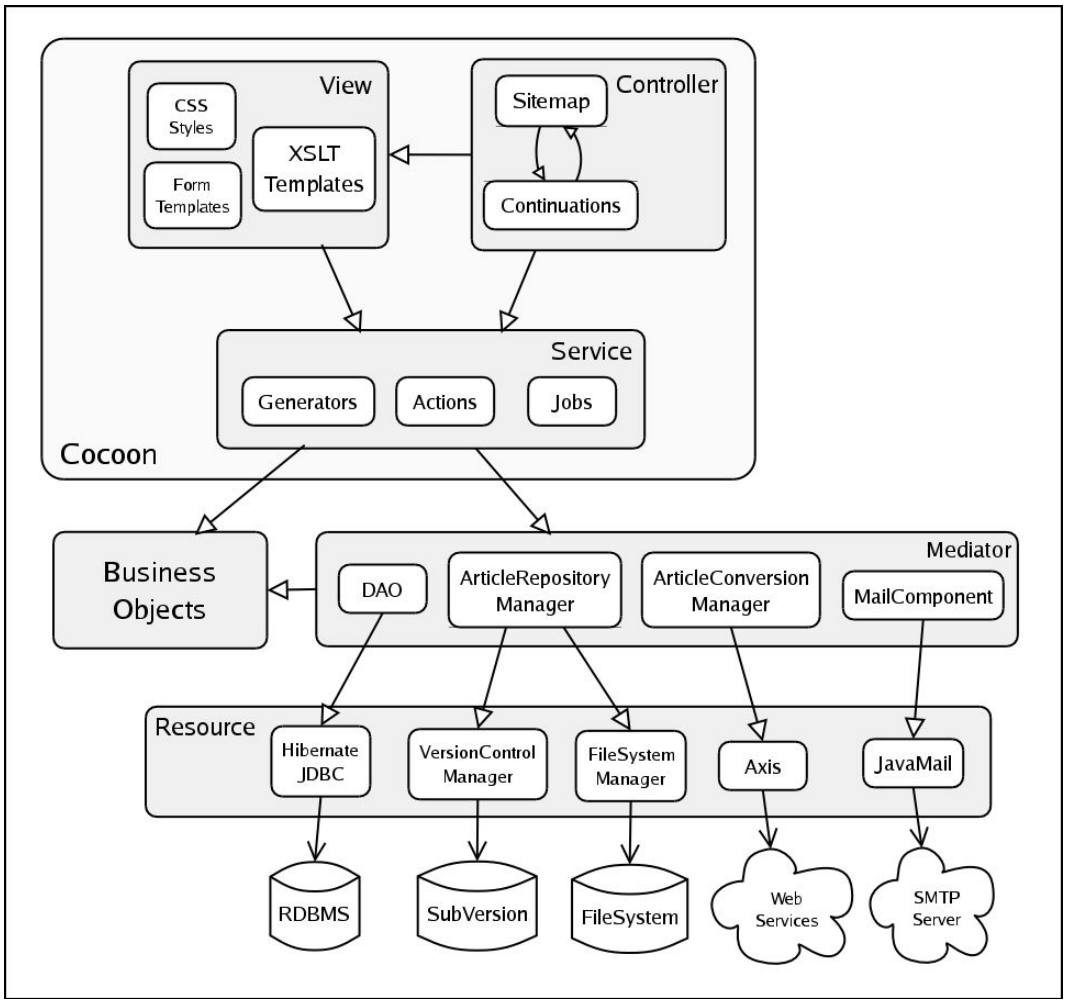
Dzięki akademickiemu charakterowi projektu oraz braku ograniczeń, jakie zazwyczaj są związane z przedsięwzięciami komercyjnymi, wykonawcy mieli pełną swobodę w doborze najnowocześniejszych technologii i eksperymentowaniu nad architekturą serwisu. Rezultatem tych eksperymentów jest właśnie szkielet architektoniczny XWA.

### 4.1.1. Szkielet publikacji Apache Cocoon

Trzonem technologicznym serwisu e-Informatyka są technologie bazujące na platformie Java, XML oraz szkielecie Apache Cocoon [WWW2004a]. Połączenie tych technologii otwiera przed projektantem olbrzymi wachlarz możliwości, który przy pierwszym spotkaniu może sprawiać jednak wrażenie chaosu. Wrażenie to jest potęgowane poprzez brak dokumentów opisujących zalecenia i dobre praktyki dotyczące projektowania aplikacji na bazie szkieletu Apache Cocoon. Niniejszy rozdział jest próbą uzupełnienia tej luki.

Rozwiązaniem, na którym projekt od początku bazował jest szkielet publikacji Apache Cocoon. Jednak architektura systemu została w znacznym stopniu przeprojektowana pod koniec 2003 roku. Kluczowe elementy nowej architektury serwisu znalazły swoje miejsce w przedstawionym szkielecie architektonicznym XWA.

Cocoon sam w sobie jest szkieletem publikacji wykorzystującym koncepcje architektury potokowej (ang. *pipe architecture*). W związku z tym już samo odwzorowanie takiego szkieletu na architekturę warstwową może być dyskusyjne, a co za tym idzie jest sporym wyzwaniem. Zaproponowana przez autorów architektura i podział komponentów szkieletu Cocoon na poszczególne pakiety jest wynikiem doświadczeń autorów. Uwzględnia jednak interesujące spostrzeżenia oraz sugestie ekspertów i konsultantów ze środowisk korporacyjnych.




Rys. 5. Architektura serwisu e-Informatyka

Należy zauważyć, że Apache Cocoon jest narzędziem wyjątkowo elastycznym, a zarazem potężnym. Oferuje niezwykle szerokie spektrum możliwości rozwiązania

każdego problemu. Jest to szczególnie widoczne, gdy wykorzysta się wbudowaną architekturę komponentową (szkielet Avalon) umożliwiającą tworzenie aplikacji zgodnie z metaforą klocków Lego.

Architekturę serwisu e-Informatyka przedstawia rysunek 5., natomiast szczegóły implementacyjne związane z poszczególnymi pakietami zostały opisane w dalszej części tej sekcji.

### 4.1.2 Dwójka z triady (View-Control)

Serwis e-Informatyka w warstwie prezentacji intensywnie wykorzystuje technologie bazujące na XML. Związane z nimi zasoby są zlokalizowane w pakiecie **View**. Są to przede wszystkim dokumenty XML określające logiczną zawartość portalu oraz transformaty XSL (*Extensible Stylesheet Language*) opisujące przekształcenia wcześniejszych dokumentów do postaci HTML.

Zadaniem pakietu **Controller** jest sterowanie przepływem w ramach interakcji użytkownika z systemem (wywołanie akcji z logiki aplikacji i uruchomienie generacji widoku).

Kontroler odwzorowuje (ang. *mapping*) żądania użytkownika (ang. *request*) na odpowiednie wywołania logiki aplikacji lub zmianę widoku. Zazwyczaj odwzorowania bazują na żądanym przez użytkownika adresie URL. Mogą również uwzględniać inne parametry, takie jak profil użytkownika i urządzenia dostępowego (ang. *user agent*) umożliwiając personalizację widoku [Made2002].

Funkcje kontrolera w szkielecie Cocoon pełni **mapa serwisu** (ang. *sitemap*). Jest to dokument XML, który zawiera odwzorowania żądań (ang. *request*) użytkownika na odpowiednie potoki (ang. *pipeline*). Precyzują one proces generowania widoku opisując sekwencje przekształceń dokumentów XML. W czasie tego procesu mogą być wywołane akcje systemu (z pakietu **Service**).

Na rysunku 6. przedstawiono przykładowy fragment mapy serwisu odpowiadający za wygenerowanie listy recenzji danego artykułu w postaci strony HTML.

```
1   <map:pipeline>
2     <map:match pattern="show-review/*">
3       <map:generate type="article-review">
4         <map:parameter name="article-id" value="{1}"/>
5       </map:generate>
6       <map:transform type="xsl" src="xsl/reviewlist2html xslt"/>
7       <map:transform type="xsl" src="xsl/site2html xslt"/>
8       <map:serialize type="html"/>
9    </map:match>
10  </map:pipeline>
```

Rys. 6. Przykładowy fragment mapy serwisu

Znacznik `<map:match>` (linia 2) określa wzorzec, według którego będą dopasowywane adresy URL. Jeżeli żądany adres spełnia podany wzorzec, wtedy realizowany jest potok. Rozpoczyna go generator (linia 3), który produkuje dokument XML (w postaci strumienia zdarzeń SAX) zawierający listę recenzji. Generator przyjmuje jako parametr identyfikator artykułu (linia 4), którego recenzje mają zostać wyświetlone. Następnie

dokument jest kolejno przekształcany przez transformaty XSL. Pierwsza z nich (linia 6) przekształca listę recenzji w formacie XML do postaci HTML, druga (linia 7) osadza wynik poprzedniej w standardowym wyglądzie portalu. Potok kończy serializator (ang. *serializer*) zamieniający strumień XML na postać wynikową (np. HTML, PDF, WML). Szersze omówienie standardowych mechanizmów działania Apache Cocoon można znaleźć między innymi w [Zieg2003, Made2003b].

Przykład doskonale ilustruje modularność systemów bazujących na XML. Ponowne użycie elementów potoku jest wyjątkowo proste. Na przykład transformata osadzająca artykuł w wyglądzie portalu, może zostać również wykorzystana do osadzenia dowolnego innego dokumentu.

W przypadku zaawansowanych aplikacji internetowych, wymagających implementacji sekwencji interakcji, pojawia się konieczność utrzymywania stanu realizacji interakcji (aktualnej pozycji w sekwencji interakcji) użytkowników z systemem. Implementacje tego zadania można w znacznym stopniu uprościć wykorzystując koncepcję **kontynuacji** (ang. *continuations*) i wprowadzając **kontroler kontynuacji**. Pozwoli on przesłonić mechanikę protokołu HTTP i przyspieszyć w znacznym stopniu pracę nad rozwojem aplikacji internetowej.

W Apache Cocoon kontroler kontynuacji jest realizowany za pomocą mechanizmu organizującego przepływ sterowania (ang. *control flow*) [WWW2004b]. Mechanizm ten może być realizowany w różny sposób np. implementowany za pomocą skryptów modelujących przepływ (ang. *flowscript*) lub w języku Java.

Na rysunku 7. przedstawiony został prosty przykład ilustrujący koncepcję *flow*. Jest to wielokrokowy formularz rejestracji użytkownika.

```
1  function register() {
2    sendPageAndWait("user-form");
3    var user = cocoon request get("user");
4    sendPageAndWait("address-form");
5    var address = cocoon request get("address");
4    sendPageAndWait("password-form");
5    var password = cocoon requestg et("password");
6    sendPageAndWait("register-confirm");
9    AccountServicer egister(user, address, password);
10   sendPage("register-successful");
11 }
```

Rys. 7. Przykład skryptu *flowscript*

Przebieg konwersacji jest następujący. Użytkownik krokowo wypełnia kolejne formularze. Podaje swoje dane osobowe, adresowe, hasło i ostatecznie potwierdza wprowadzone dane. System w odpowiedzi potwierdza zarejestrowanie użytkownika. Warto podkreślić fakt, że każdy krok jest przedstawiony użytkownikowi jako osobna strona WWW.

Wywołania funkcji `sendPageAndWait` są kluczowymi punktami tego przykładu. Wykonują one wewnętrzne żądanie do wybranego adresu URL. Przesyłają wygenerowaną stronę użytkownikowi i następnie oczekują na jego reakcję.

Równocześnie stan aplikacji jest zapisywany. W momencie, gdy użytkownik odpowie, realizacja skryptu jest kontynuowana od miejsca, w którym została przerwana.

### 4.1.3 Ostatni z triady (Model)

Projekt architektoniczny e-Informatyki intensywnie wykorzystuje wbudowaną w Apache Cocoon architekturę komponentową (bazując na szkielecie Avalon). Cała logika aplikacji zawarta w pakiecie **Service** jest oparta właśnie na interfejsach komponentowych. Klasy zawarte w tym pakiecie można podzielić na trzy grupy: generatory (ang. *generators*), akcje (ang. *actions*) oraz zadania (ang. *jobs*).

**Generatory** są komponentami ogólnego przeznaczenia, produkującymi dokumenty XML w postaci zdarzeń SAX. Ich najczęstszym zadaniem jest udostępnienie warstwie prezentacji danych zawartych w obiektach biznesowych. Na przykład generowanie listy recenzji ostatnio zgłoszonych artykułów. Główną zaletą generatorów jest przysłonięcie szczegółów realizacji obiektów biznesowymi (typowa *Facade* [Gamm1995]), jak również wyrażenie danych biznesowych w wyjątkowo elastycznym języku jakim jest XML.

**Akcje** są to komponenty implementujące logikę aplikacji. Centralizują one wywołania logiki biznesowej, dostępu do zewnętrznych źródeł danych, czy mechanizmów trwałości danych dotyczących jednego przypadku użycia. Dobrym przykładem jest akcja obsługująca zgłoszenie artykułu składająca się z wywołania zapisu do bazy danych informacji o przesłaniu nowego artykułu, a także zapamiętania treści artykułu w repozytorium. Trzeba podkreślić, że akcje nie zawierają realizacji powyższych operacji. Ich szczegóły znajdują się w warstwach niższych. Akcje wywołują je tylko spinając w transakcje (ang. *user transactions*). Są typowymi realizacjami wzorca *Application Service* [Alur2003] tylko o większym rozdrobnieniu interfejsu, co jest spowodowane specyfiką szkieletu Apache Cocoon.

**Zadania** są analogicznymi komponentami do akcji. Różnica polega na tym, że zadania są wykonywane przez wbudowany harmonogram, na przykład co dwa dni, a nie na żądanie użytkownika (tak jak akcje). Przykładem zadania jest usuwanie konta użytkownikom, którzy nie aktywowali go po tygodniu.

Pakiet **Business Objects** zawiera klasy będące obiektami biznesowymi. W przypadku projektu e-Informatyka są to głównie klasy posiadające jedynie aspekt statyczny, a co za tym stoi, trwałość (ang. *persistence*) tych obiektów odgrywa tu główną rolę.

Pakiet **Mediator** pośredniczy w dostępie do szerokiego spektrum zewnętrznych źródeł danych, których obsługa jest zaimplementowana w pakiecie **Resource**. Ilustruje to rysunek 5. Trwałość obiektów zapewnia system mapowania obiektowo relacyjnego Hibernate. W warstwie pośredniczącej są komponenty implementujące wzorzec DAO. Innym przykładem jest obsługa repozytorium treści, które fizycznie implementuje system zarządzania wersjami.

# 5. Podsumowanie

Bazując na wzorcach MVC i PCMEF zaproponowano szkielet architektoniczny XWA uwzględniający specyfikę nowoczesnych aplikacji internetowych. W szczególności:

- Precyzyjnie wyspecyfikowano semantykę poszczególnych pakietów i warstw w kontekście aplikacji internetowych, a w szczególności technologii platformy Java oraz szkieletu Apache Cocoon.
- Zapewniono, bardzo potrzebną w przypadku aplikacji internetowych, wyraźną i klarowną separację widoku, kontrolera i modelu, definiując odpowiedzialność elementów triady w kontekście aplikacji internetowych.
- Wyodrębniono i omówiono potrzebę wprowadzenia nowego elementu architektury – kontrolera kontynuacji. Wprowadzenie tego elementu umożliwia tworzenie aplikacji internetowych, jak klasycznych programów, co pozwala na znaczne uproszczenie implementacji skomplikowanych interakcji użytkownika z systemem.
- Omówiono nowy rodzaj kontraktów między warstwami. Bazuje on na interfejsach XML wyrażonych poprzez schematy dokumentów XML (np. za pomocą XML Schema), a nie na klasycznych interfejsach np. znanych z języka Java. W przypadku architektury XWA dobrym przykładem interfejsu XML jest interfejs między logiką aplikacji a widokiem.

Warto zauważyć, że XWA jest w założeniu architekturą rozszerzalną dzięki podziałowi na warstwy (zaczerpniętemu z PCMEF) i zastosowaniu technologii komponentowych do implementacji logiki aplikacji. Zadania związane z prezentacją oraz integracją z innymi systemami realizowane są z wykorzystaniem XML, co również ma wpływ na rozszerzalność zaprezentowanego szkieletu architektonicznego.

XWA i jego praktyczna implementacja wypełniają również poważną lukę spowodowaną brakiem dokumentów opisujących dobre praktyki dla szkieletu publikacji Apache Cocoon.

# 6. Podziękowania

Autorzy pragną wyrazić swoją wdzięczność członkom zespołu e-Informatyka za cenne spostrzeżenia, a prof. Zbigniewowi Huzarowi za motywację do napisania tego rozdziału.

# Bibliografia

[Alur2003] Alur D., Crupi J., Malks D., *Core J2EE Patterns: Best Practices and Design Strategies,* Prentice Hall Ptr, 2003.

[Burb1987] Burbeck S., *Applications Programming in Smalltalk-80(TM): How to use Model-View-Controller (MVC)*, 1987, http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html

[Busc1996] Buschmann F., et al., *Pattern-Oriented Software Architecture, Volume 1: A System of Patterns*, John Wiley & Sons, 1996.

[Gamm1995] Gamma E., Helm R., Johnson R., Vlissides J., *Design Patterns, Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[Fowl1998] Fowler A., *A Swing Architecture Overview – The Inside Story on JFC Component Design,* http://java.sun.com/products/jfc/tsc/articles/architecture/

[Fowl2003] Fowler M., *Patterns of Enterprise Application Architecture*, Addison-Wesley, 2003,

[Mac2003a] Maciaszek L.A., Liong B.L, *Designing Measurably-Supportable Systems*, Nowoczesne Technologie Informatyczne w Zarządzaniu, Red. E. Niedzielska, H. Dudycz, M. Dyczkowski, Prace Naukowe Akademii Ekonomicznej we Wrocławiu 2003 nr 986, s. 120-149.

[Mac2003b] Maciaszek L.A, *Znaczenie architektonicznego projektu systemu w inżynierii oprogramowania*, Problemy i Metody Inżynierii Oprogramowania, Red. Z. Huzar, Z. Mazur, Wydawnictwa Naukowo-Techniczne, 2003, s.15-23.

[Maci2004] Maciaszek L.A., Liong B.L., Bills S., *Practical Software Engineering, A Case-Study Approach*, Addison-Wesley, 2004.

[Made2002] Madeyski L., *Nowe koncepcje tworzenia aplikacji internetowych na przykładzie portalu e-informatyka.pl,* Nowoczesne Technologie Informatyczne w Zarządzaniu, Red. E. Niedzielska, H. Dudycz, M. Dyczkowski, Prace Naukowe Akademii Ekonomicznej we Wrocławiu 2002 nr 955, s. 425-437.

[Made2003a] Madeyski L., Kubasiak M., *Zwinna specyfikacja wymagań,* Problemy i Metody Inżynierii Oprogramowania, Red. Z. Huzar, Z. Mazur, Wydawnictwa Naukowo-Techniczne, 2003, s.53-68.

[Made2003b] Madeyski L., Mazur P., *Nowoczesne aplikacje internetowe*, Telenet Forum, maj 2003, s. 14-17.

[Sesh1999] Seshadri G., *Understanding JavaServer Pages Model 2 architecture*, http://www.javaworld.com/javaworld/jw-12-1999/jw-12-ssj-jspmvc.html

[WWW2004a] The Apache Cocoon Project, http://cocoon.apache.org

[WWW2004b] The Apache Cocoon - Flow, http://cocoon.apache.org/2.1/userdocs/flow/

[Zieg2002] Ziegeler C., Langham M., *Cocoon: Building XML Applications*, New Riders 2002.