Zastosowanie technologii Web Farming dla poprawy procesu wytwarzania oprogramowania

Grzegorz Cysewski, Krzysztof Goczyła
Wydział Elektroniki, Telekomunikacji i Informatyki, Politechnika Gdańska
{grechut, kris}@eti.pg.gda.pl

Streszczenie

Web Farming jest technologią mającą na celu odkrywanie źródeł informacji w sieci WWW, zbieranie odkrytych informacji, strukturalizowanie ich i rozpowszechnianie do odpowiednich odbiorców. W rozdziale zaprezentowano koncepcję wykorzystania technologii Web Farming w sieci intranet firmy produkującej oprogramowanie w celu poprawy procesu wytwórczego. Przedstawiono ogólną architekturę systemu (o nazwie So-Far) oraz sposób jego wykorzystania przez różne grupy udziałowców procesu wytwarzania oprogramowania. Zdefiniowano język opisu procesu wytwarzania oprogramowania oraz model obiektowy procesu, na którym bazuje ten język.

Wprowadzenie

Wytwarzanie oprogramowania jest złożonym procesem składającym się z etapów, w których kolejni udziałowcy procesu dodają coraz bardziej szczegółową informację dotyczącą budowanego systemu. Aby proces ten mógł być zakończony sukcesem, tj. wytworzeniem systemu zgodnego z wymaganiami użytkownika, musi być spełnionych szereg wymagań związanych ze środowiskiem wytwórczym oraz efektywnością pracy zespołu. Każda organizacja tworząca oprogramowanie tworzy swoje własne środowisko, zależne od rozmiaru organizacji i jej zasobów, a także od rozmiaru budowanego systemu. Z punktu widzenia koncepcji prezentowanej poniżej interesują nas środowiska wytwórcze dla średnich i dużych projektów informatycznych. W średnich projektach często wykorzystywane są narzędzia CASE (ang. *Computer Aided Software Engineering*), wspierające wszystkie lub wybrane fazy procesu wytwarzania oprogramowania. W dużych projektach wykorzystywane są narzędzia typu CSCW (ang. *Computer Supported Cooperative Work*), tworzące środowisko składające się z narzędzi wersjonujących, dokumentujących i innych, o charakterze środków wspomagających.

Z reguły im większy jest projekt, tym bardziej rygorystyczne zasady współpracy muszą być przestrzegane przez jego uczestników [BAY1999, COHU1995, GOR2000, SZEJ2002]. Zasady te obejmują zadania dokumentacyjne, które muszą być regularnie wykonywane przez członków zespołu. Dokumentacja ta musi być regularnie wstawiana do repozytorium projektu. W ten sposób prowadzący projekt może w każdej chwili śle-

dzić postęp prac i ocenić stopień zaawansowania prac. Niestety, zadania dokumentacyjne zazwyczaj zwiększają obciążenie członków zespołu, co więcej, są przez niektórych z nich traktowane jako zbędne i nadmiarowe. W rezultacie repozytorium projektu pozostaje niekompletne i nieaktualne.

Problem ten dotyczy nie tylko dokumentacji projektu jako całości, ale również poszczególnych komponentów systemu (klas, funkcji), które mogłyby zostać ponownie użyte w tym samym lub w innych projektach. Zwykle organizacje tworzą wydzielone repozytorium takich komponentów. Jednakże w takim rozwiązaniu istnieje problem zapełniania repozytorium; zazwyczaj istnieje wydzielony administrator, który jest odpowiedzialny za wyszukiwanie nowych komponentów i wstawianie ich do repozytorium. Stanowi to duży problem, gdyż programiści i projektanci często nie trzymają się standardów wymaganych przez repozytorium komponentów ponownego użycia - komponenty nie są tworzone pod kątem ich ponownego użycia (ang. *for reuse*). Dodatkowo wytwórcy komponentów często nie uaktualniają repozytorium po zmianach dokonanych w komponentach. A zatem i takie repozytorium jest często niekompletne i nieaktualne.

Trzecim problemem, z którym borykają się organizacje wytwarzające oprogramowanie, jest dostarczanie informacji dla pionu zarządzającego. Menedżerowie organizacji nie oczekują szczegółowych informacji na temat projektu, a jedynie informacji dotyczących postępu prac, zagrożeń oraz przewidywanego terminu zakończenia projektu. Centralne repozytorium takich strategicznych informacji musi być regularnie zasilane aktualnymi informacjami przez kierowników projektów i - w efekcie - wykazuje te same wady, co repozytoria wymienione powyżej.

Niniejszy rozdział przedstawia podejście, które niweluje wymienione, a także inne, problemy związane z zarządzaniem procesem wytwarzania oprogramowania. W podejściu tym, informacje potrzebne na różnych poziomach realizacji projektu (wytwórca, kierownik projektu, menedżer organizacji) pozostają rozproszone w sieci intranet organizacji i są udostępniane dla procesu Web Farming. Proces ten zbiera potrzebne informacje z serwerów źródłowych i udostępnia je użytkownikom. W ten sposób znikają dwa główne problemy - po pierwsze: nie ma potrzeby uzupełniania repozytorium przez członków zespołu wytwórczego i po drugie: informacje zebrane w ten sposób są zawsze aktualne.

Web Farming w procesie wytwarzania oprogramowania

Web Farming jest technologią, która umożliwia zbieranie i udostępnianie informacji z sieci WWW [HACK1999]. Ogólna idea technologii Web Farming obejmuje zaplanowanie i implementację w organizacji procesu składającego się z niżej opisanych czterech faz.

- *Discovery (odkrywanie)* .

  W tej fazie („oranie pola") sieć WWW jest przeszukiwana w celu odnalezienia źródeł zawierających informacje istotne z punktu widzenia danej organizacji. W procesie Web Farming przebiegającym w globalnej sieci jest to najtrudniejsza faza, wymagająca najwięcej ingerencji człowieka. Jednak w procesie Web Farming ogra-

niczonym do lokalnego intranetu organizacji, faza ta może zostać całkowicie zautomatyzowana, gdyż zbiór przeszukiwanych źródeł jest ograniczony, a ponadto mogą być przyjęte ścisłe założenia dotyczące struktury i formatu wyszukiwanych informacji. W procesie wytwarzania oprogramowania serwerami źródłowymi są repozytoria projektów (np. repozytoria narzędzi CASE) oraz lokalne węzły poszczególnych członków zespołu.

- *Acquisition (zbieranie)*.

  Podczas tej fazy („sianie") informacje odkryte w poprzedniej fazie są zbierane i przechowywane w repozytorium procesu. Fazę tę znacznie łatwiej zautomatyzować, nawet w procesie Web Farming przebiegającym w sieci globalnej. Ważną kwestią tej fazy jest zachowywanie informacji historycznych, zebranych wcześniej, szczególnie w aspekcie zbierania informacji o procesie wytwarzania oprogramowania, np. w celu monitorowania postępów w pracach.

- *Structuring (strukturalizacja)*.

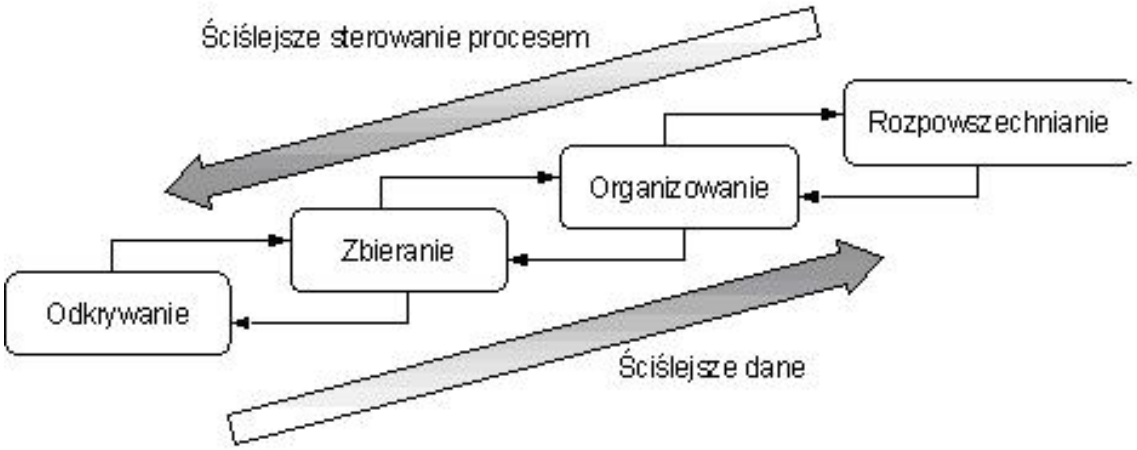
  W trakcie tej fazy („pielęgnowanie zasiewów") zebrane informacje są analizowane, walidowane i transformowane do formatu wymaganego przez użytkowników („konsumentów" informacji). Błędy, braki oraz powtórzenia danych powinny zostać wykryte i usunięte. Końcowy format informacji zależy od grupy użytkowników będącej odbiorcą informacji (wytwórca, kierownik projektu, menedżer organizacji), stąd też faza ta musi być bardzo elastyczna i w pełni parametryzowalna. Strategia Web Farming zakłada, że dane zebrane w poprzednich fazach powinny zostać ustrukturalizowane w postaci składnicy (hurtowni) danych będącej źródłem dalszych analiz [KIMB1996].

- *Dissemination (rozpowszechnianie)*

  Końcowa faza procesu Web Farming („zbieranie plonów") polega na udostępnieniu danych odbiorcom (ludziom lub repozytoriom zewnętrznym). Docelowo dane mogą zostać poddane dodatkowym analizom przy użyciu specjalnych narzędzi prezentacyjnych oraz narzędzi typu *data mining* lub *business inteligence* [BTS1999, KIMB1996]. W opisywanym tu procesie Web Farming, zbiór odbiorców informacji jest ograniczony, zatem nie jest wymagany żaden uniwersalny mechanizm.

Rysunek 1. prezentuje przebieg procesu Web Farming. Im wyższy poziom fazy, tym wyższa jakość informacji. Im niższy poziom fazy, tym wymagane jest staranniejsze (precyzyjniejsze) kontrolowanie procesu.

Technologia Web Farming proponuje również narzędzia do wykorzystania w każdej z faz: przeglądarki oraz agenci w fazie *discovery* i *acquisition* oraz narzędzia do transformacji i analizy danych, hurtownie danych i narzędzia zarządzania wiedzą w fazie *structuring* i *dissemination*.



Cztery fazy procesu Web Farming

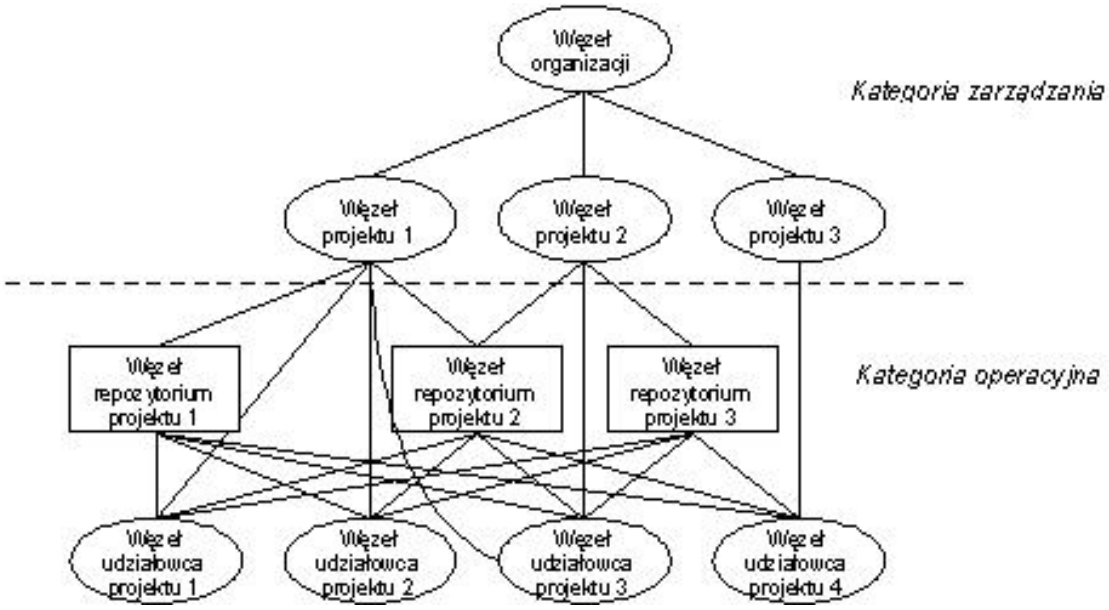
W procesie Web Farming stosowanym w wytwarzaniu oprogramowania agenci działający w sieci WWW są najbardziej użyteczni w fazie *discovery* i *acquisition*. Agenci lokalni, umieszczani na pojedynczych stacjach roboczych, mogą inicjować fazę zbierania informacji we współpracy z innymi agentami lokalnymi działającymi na innych stacjach. Bez trudu mogą również funkcjonować specjalizowani agenci zdalni, aktywnie penetrujący sieć, gdyż wewnątrz organizacji będą oni traktowani jako agenci zaufani. Dla faz *structuring* i *dissemination* wymagane są specjalizowane narzędzia, współpracujące z narzędziami CASE używanymi w danej organizacji.

Struktura informacji

Proces Web Farming zastosowany w wytwarzaniu oprogramowania (nazywany dalej *SoFar*, od ang. *Software Farming*) działa na informacjach istniejących w lokalnym intranecie organizacji. Zakłada się, że formatem analizowanych informacji jest XML lub HTML (inne formaty przed analizą zostaną poddane transformacji do XML).

Zwykle w projekcie informatycznym informacje podzielone są na szereg logicznych kategorii (węzłów logicznych), które mogą być postrzegane jako producenci i/lub konsumenci informacji w procesie Web Farming. Węzły te mogą być następujących typów (por. rysunek 2):

- węzeł organizacji (konsument),
- węzeł projektu (producent/konsument),
- węzeł repozytorium projektu (producent),
- węzeł udziałowca projektu (producent/konsument).



Klasy informacji w procesie \textit

Węzeł organizacji zawiera informacje o wszystkich projektach realizowanych przez organizację, ich aktualnym statusie i postępie w pracach. Węzeł ten pobiera dane z węzłów projektów.

Węzeł projektu zawiera kompletne informacje o pojedynczym projekcie - zadania, zasoby, budżet itp. Dostarcza dane dla węzła organizacji i pobiera informacje dostarczane przez węzły repozytoriów projektów oraz węzły udziałowców projektów.

Węzeł repozytorium projektu zawiera informacje pochodzące ze zdefiniowanego repozytorium projektu oraz repozytoriów narzędzi używanych w procesie wytwórczym (CASE, CSCW). Węzeł ten dostarcza informacje dla węzłów projektów oraz węzłów udziałowców projektów.

Węzeł udziałowca projektu jest węzłem wytwórcy (analityka, projektanta, programisty, testera) zaangażowanego w dany projekt i zawiera informacje wytworzone przez niego na rzecz projektu. Węzeł ten pobiera informacje z wszystkich węzłów repozytoriów projektów, zaś dostarcza informacje dla węzłów tych projektów, w które jest zaangażowany dany wytwórca.

Węzły mogą zostać zaklasyfikowane jako aktywne lub pasywne.

Węzeł aktywny (reprezentowany na rysunku 2. jako owal) może inicjować proces *SoFar*.

Węzeł pasywny (prostokąt) jest tylko producentem danych przeznaczonych dla węzłów aktywnych.

Węzeł aktywny może znajdować się również w stanie pasywnym - wówczas jest producentem danych dla aktualnie aktywnych węzłów. Klasa węzła (aktywny/pasywny) jest związana z jego rolą (konsument/producent), tzn. węzeł jest w stanie aktywnym, jeśli konsumuje dane, a jest w stanie pasywnym, jeśli je produkuje.

Węzły zostały podzielone na dwie kategorie:

- kategoria zarządzania,
- kategoria operacyjna.

Węzły kategorii zarządzania są istotne z punktu widzenia menedżera organizacji i zawierają informacje pobrane bezpośrednio przez menedżerów organizacji i projektów. Węzły kategorii operacyjnej są używane przez wytwórców systemu i dostarczają dane dla węzłów kategorii zarządzania.

Rezultatem procesu *SoFar* są informacje zbierane na trzech poziomach abstrakcji:

- na poziomie menedżera organizacji,
- na poziomie kierownika projektu,
- na poziomie wytwórcy.

Poziom menedżera organizacji zawiera informacje zebrane przez menedżerów organizacji, którzy chcą mieć ogólny wgląd na stan projektów. Informacje te to zużywane zasoby, aktualny status, koszty, szacowany nakład pracy itp. Informacje z tego poziomu zasilają węzeł organizacji.

Poziom kierownika projektu zawiera informacje o stanie konkretnego projektu z punktu widzenia jego kierownika. Informacje te obejmują dane na temat statusu każdego ze zdefiniowanych zadań, harmonogramu, zasobów projektu itp. Informacje z tego poziomu zasilają węzły projektów.

Poziom wytwórcy zawiera informacje użyteczne dla wytwórcy zaangażowanego w projekt, czyli modele, projekty i kody źródłowe, komponenty oraz inne elementy ogólnie pojętego know-how. Informacje z tego poziomu zasilają węzły udziałowców projektów.

Poziomy abstrakcji określają również poziomy bezpieczeństwa. Dla zachowania bezpieczeństwa potrzebna jest identyfikacja użytkowników, gdyż nie wszystkie informacje o projektach powinny być dostępne dla każdego. Przyjmuje się, że im wyższy poziom abstrakcji, tym szerszy dostęp do informacji, według następujących zasad:

- członkowie poziomu menedżera organizacji mają dostęp do wszystkich danych (z ograniczeniami ich modyfikacji),
- członkowie poziomu kierownika projektu mają dostęp do danych dotyczących ich własnego projektu (z ograniczeniami ich modyfikacji),
- członkowie poziomu wytwórcy mają pełen dostęp do swoich własnych węzłów.

Jak widać, członkowie poziomów (czyli grup użytkowników systemu) powinni mieć specjalne prawa do dostarczania (zapis) oraz pobierania (odczyt) danych z węzłów logicznych - stąd potrzeba zdefiniowania ról, jakie będą pełnić użytkownicy w ramach grup.

Architektura systemu

Architektura systemu *SoFar* (rysunek 3) jest oparta na rozproszonym procesie Web Farming opisanym w podrozdziale 2. Każdy węzeł zawiera informacje, które są zbierane i analizowane przez system *SoFar* zgodnie z zasadami procesu Web Farming.

- *Odkrywanie*.

  Ta faza jest oparta na intranecie organizacji. Każdy komputer przyłączony do sieci może zostać wytypowany jako węzeł-kandydat, na którym poszukiwane są informacje istotne z punktu widzenia procesu. Rodzaje węzłów są niezależne od roli, jaką pełnią komputery w sieci - wszystkie węzły mogą znajdować się na jednej fizycznej maszynie lub być rozproszone w sieci intranet organizacji. Najważniejszym zadaniem tej fazy jest sklasyfikowanie znalezionej informacji i przypisanie do logicznych węzłów.

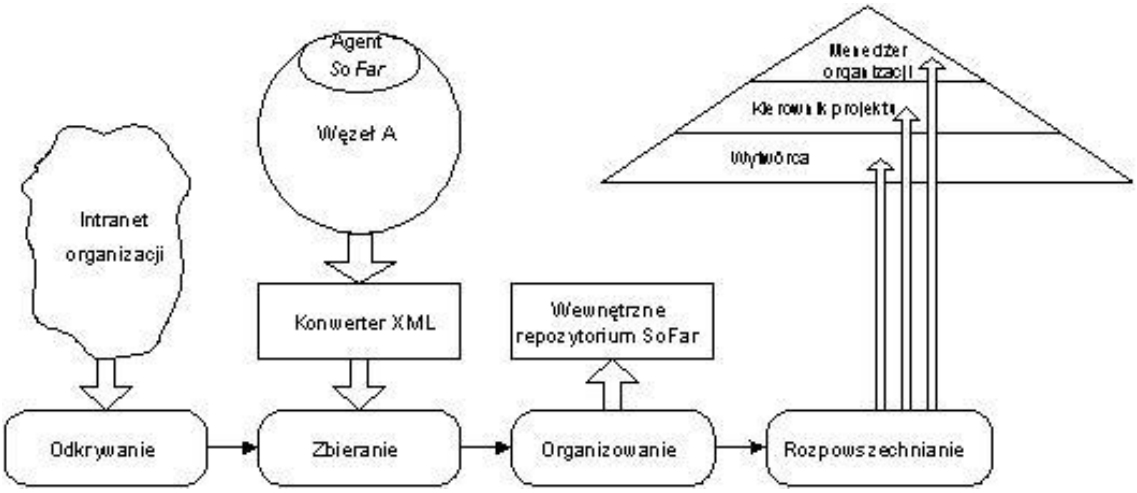
- *Zbieranie*.

  W tej fazie są używani specjalizowani lokalni i zdalni agenci, którzy aktywizowani są przez węzeł aktywny (czyli konsument) na węzłach pasywnych (czyli na producentach). Zebrana informacja jest przekazywana poprzez sieć. Agenci mogą pracować w sposób ciągły, zatem proces *SoFar* może być uruchomiony w tle innych zadań organizacji. Założeniem jest, iż informacja pobrana w tej fazie występuje w formacie XML; jeśli nie, to zakłada się, iż istnieje możliwość transformacji danych do formatu XML. Obecnie wszystkie narzędzia typu CASE umożliwiają taką transformację. Możliwe jest również opracowanie własnych procedur transformacji dedykowanych systemowi *SoFar*.

- *Organizowanie*. Podczas tej fazy informacje są dzielone na trzy poziomy abstrakcji opisane powyżej i przechowywane w wewnętrznym repozytorium, które umożliwia prezentację danych w sposób użyteczny dla końcowych odbiorców. W ostatnim czasie producenci systemów baz danych (Oracle, IBM, Microsoft) udostępniają rozszerzenia swoich systemów umożliwiające stosunkowo łatwe manipulowanie dokumentami w formacie XML, co ułatwia budowę takiego repozytorium. Najważniejsza część informacji przetwarzanych w procesie *SoFar* jest uzyskiwana z węzłów repozytoriów projektów. W podejściu *SoFar* dodatkowym założeniem jest to, iż proces wytwórczy oparty jest na metodologii obiektowej [RAT2002] oraz języku UML. Założenie to umożliwia przedstawienie modeli UML zawartych w repozytoriach projektów w formacie XML za pomocą jednej z istniejących definicji XML dedykowanych modelom UML. Możliwe jest tu użycie na przykład standardu *OMG XML Metadata Interchange* (XMI) [OMG2002], którego głównym celem jest wymiana danych pomiędzy różnymi narzędziami do modelowania.

- *Rozpowszechnianie*.

  Celem tej fazy jest zaprezentowanie odpowiednich danych w odpowiednim czasie odpowiednim odbiorcom. W tej fazie informacje z repozytorium procesu *SoFar* są przekazywane odpowiednim konsumentom. Ważnym aspektem tej fazy jest struktura rozpowszechnianych informacji: muszą one być prezentowane w sposób jak najbardziej zbliżony do przyjętych w organizacji standardów.



Architektura logiczna systemu SoFar

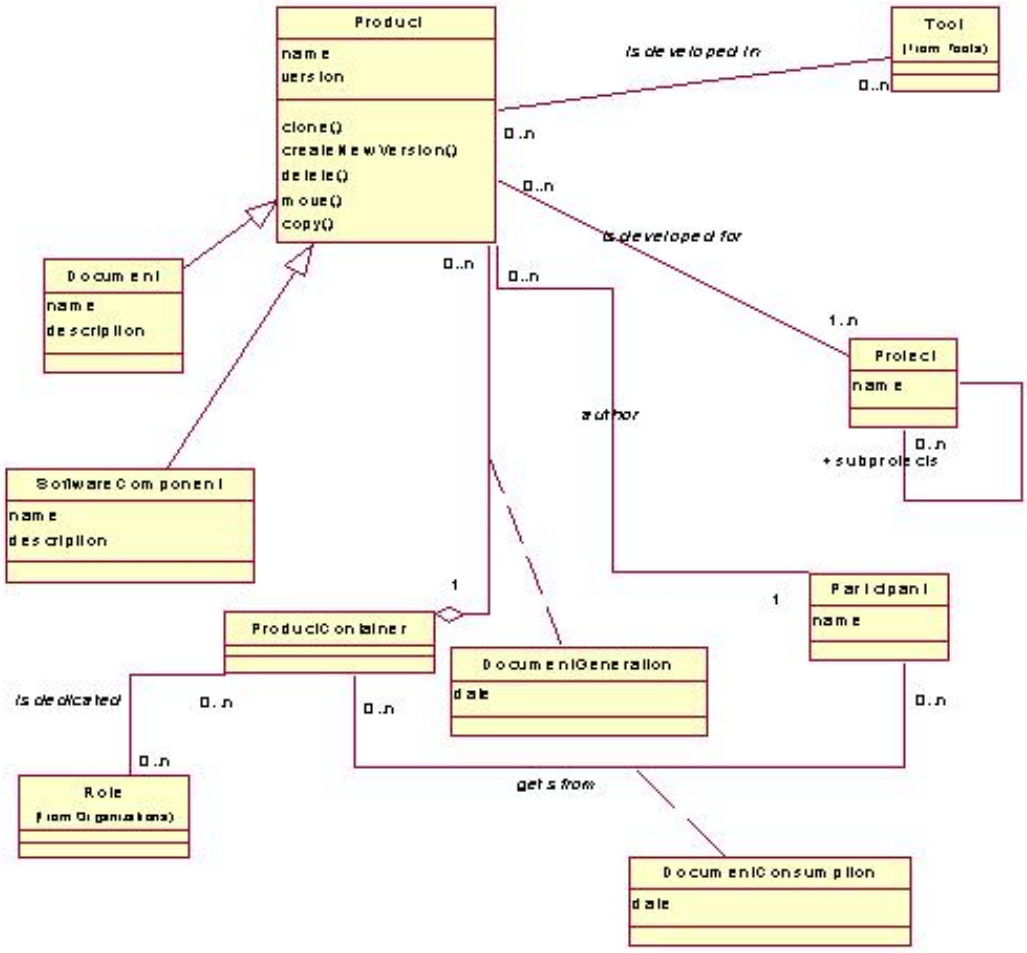
Model obiektowy procesu wytwórczego

Proces wytwórczy w systemie *SoFar* został opisany za pomocą modelu obiektowego. Każdy proces jest instancją ogólnego opisu wyrażonego za pomocą diagramu klas, zaprezentowanego na rysunku 4.

Zakłada się, że projekt informatyczny można opisać za pomocą sekwencji wytwarzania kolejnych produktów składających się na wynik projektu, którym jest system informatyczny.

Produkt (klasa *Product*) może być dokumentem tekstowym, kodem źródłowym, modelem UML itp. Proces wytwórczy polega na przekazywaniu przez uczestników projektu (klasa *Participant*) produktów do miejsc ich udostępnienia dla dalszego ich przetwarzania (np. rozwoju) lub też dla powstania kolejnych produktów (np. faza analizy kończy się wytworzeniem serii produktów, które po udostępnieniu, czyli opublikowaniu, są podstawą fazy projektu szczegółowego). Miejsca udostępniania (publikowania) produktów nazwano pojemnikami (klasa *ProductContainer*). Ogólnie, każdy proces wytwórczy w systemie *SoFar* może zostać opisany jako:

- zbiór uczestników procesu (ożywionych lub nieożywionych),
- zbiór pojemników,
- zbiór produktów,
- zbiór przepływów produktów pomiędzy pojemnikami, realizowanych przez uczestników we współbieżnych wątkach.



Model obiektowy procesu wytwórczego w systemie \textit - pakiet \textit

Pojemniki mogą znajdować się na różnych poziomach abstrakcji procesu - od pojemnika uczestnika projektu służącego do przekazywania produktów pomiędzy różnymi członkami zespołu wytwórczego, po repozytorium całej organizacji.

Zasadniczą kwestią dla prawidłowego zdefiniowania procesu jest określenie pojemników i ich ról w procesie oraz przepływów produktów pomiędzy pojemnikami. Celowi temu służy definiowanie procesu wytwórczego w postaci zapisu w specjalnie do tego przeznaczonym języku, będącym przedmiotem następnej części tego rozdziału.

Język opisu procesu - PMDL

Dla potrzeb opisu procesu wytwórczego opartego na pojemnikach jako na miejscach publikacji produktów wytworzonych w ramach procesu skonstruowano język PMDL - *Project Management Definition Language*. Język ten zawiera elementy wystarczające do opisu dowolnego, nawet bardzo złożonego procesu wytwórczego. Poniżej zamieszczono definicje najważniejszych elementów tego języka.

Definicje

Typ (np. typ dokumentu) definiuje się za pomocą frazy definiującej. Dla języka PMDL typami podstawowymi są wszystkie typy atomowe oraz wszystkie klasy z modelu obiektowego *SoFar*. Zmienną deklaruje się poprzez określenie nazwy oraz typu zmiennej. Stałą deklaruje się poprzez określenie nazwy oraz typu stałej. Istnieją predefiniowane stałe: TRUE, FALSE, NULL.

Każda definicja procesu wyrażona za pomocą języka PMDL powinna zawierać inicjalne definicje elementów wykorzystywanych podczas definiowania procesu. Są to na przykład następujące definicje:

- statusy dokumentów - lista możliwych wartości statusu dokumentów (może zostać zdefiniowana globalnie lub dla określonego typu dokumentu),
- role - lista ról (grup) użytkowników biorących udział w procesie,
- typy dokumentów - lita typów dokumentów występujących w procesie.

Wykonywanie operatorów języka przez role

Każdy operator jest wykonywany na rzecz konkretnego obiektu (można określić go jako metodę obiektu, na rzecz którego jest wykonywany). Dodatkowo można określić rolę, którą wykonuje dany operator lub czynność. Rola określona jako wykonawca operatora (czynności) powinna zostać wcześniej zdefiniowana w sekcji definicji inicjalnych.

Zbiór operatorów działających na pojemnikach

Zdefiniowano następujące operatory działające na pojemnikach:

- put (*Product product*) - wstawienie dokumentu (*Product*) do pojemnika,
- get (*ProductDescriptor* productDescr, *Boolean* deepAcquire) - pobranie dokumentu z pojemnika; funkcja powoduje pobranie i usunięcie dokumentu z pojemnika,
- acquire (*ProductDescriptor* productDescr, *Boolean* deepAcquire) - podgląd dokumentu z pojemnika; funkcja powoduje pobranie dokumentu bez jego usunięcia z pojemnika,
- move (*ProductDescriptor* productDescr, *ProductContainer* trgtBucket, *Boolean* deepMove) - przesunięcie dokumentu z jednego pojemnika do innego (złożenie *get* i *put*),
- copy (*ProductDescriptor* srcProductDescr, *ProductDescriptor* trgtProductDescr, *ProductContainer* trgtBucket, *Boolean* deepCopy) - skopiowanie dokumentu z jednego pojemnika do innego; powstała kopia dokumentu jest nowym dokumentem (złożenie get, create i put),
- clone (*ProductDescriptor* srcProductDescr, *ProductDescriptor* trgtProductDescr, *ProductContainer* trgtBucket) - utworzenie nowej instancji tego samego dokumentu; powstała instancja będzie posiadać wszystkie cechy źródłowego dokumentu, również będzie nabywać wszystkie cechy nabywane przez dokument źródłowy

(istnieje też zależność przechodnia: cechy nabyte przez dokument docelowy zostaną nabyte przez dokument źródłowy); autorem dokumentu sklonowanego pozostaje autor dokumentu źródłowego, przenoszone są również prawa dostępu do dokumentu; klonowania dokumentu do innego pojemnika może dokonać tylko jego autor; w tym samym pojemniku może sklonować dokument każdy, kto jest do tego uprawniony,
- delete (*ProductDescriptor* productDescr, *Boolean* deepDelete) - usunięcie dokumentu z pojemnika,
- grantRole (*RoleDescriptor* role, *Rights* rights) - nadanie praw grupie użytkowników do wykonywania operacji na pojemniku.

Zbiór operatorów na dokumentach

Operatory na dokumentach działają w obrębie pojemnika, w którym istnieje dany dokument. Zdefiniowano następujące operatory na pojemnikach:

- create (*ProductName* productName) - utworzenie dokumentu,
- createNewVersion () - utworzenie nowej wersji dokumentu,
- agregate (*ProductDescriptor* productDescr) - utworzenie lub usunięcie związku pomiędzy dokumentami w tym samym pojemniku, polegającego na tym, że dokument podrzędny jest częścią dokumentu nadrzędnego,
- reference (*ProductDescriptor* productDescr, *ProductContainer* bucket = currentBucket) - utworzenie związku pomiędzy dokumentami, polegającego na tym, że dokument nadrzędny zawiera odnośnik do dokumentu podrzędnego,
- getStatus () - pobranie statusu dokumentu; zbiór dostępnych wartości statusu jest definiowalny,
- changeStatus (*Status* status) - zmiana statusu dokumentu; zbiór dostępnych wartości statusu jest definiowalny,
- grantRole (*RoleDescriptor* role, *Rights* rights) - nadanie praw grupie użytkowników do operacji na dokumencie.

Zbiór operatorów na czynnościach

Czynności są opisane poprzez sekwencję operacji wyrażonych operatorami oraz instrukcjami warunkowymi. Zdefiniowano następujące operatory na czynnościach:

- create (*ActivityName* activityName, *ParametersList* params) - utworzenie czynności,
- grantRole (*RoleDescriptor* role, *Rights* rights) - nadanie praw grupie użytkowników do operacji na czynności,
- delete () - usunięcie czynności,
- execute (*ParametersList* params) - wykonanie zdefiniowanej czynności.
- grantRole (*RoleDescriptor* role, *Rights* rights) - nadanie praw użytkownikowi do operacji na czynności.

Instrukcje warunkowe

Zdefiniowano następujące instrukcje warunkowe:

- if (*warunek*) then ... else ... - instrukcja uzależniająca wykonanie dalszych operacji od pewnych zdefiniowanych warunków,
- while (*warunek*) ... - wykonanie iteracyjne pewnej grupy instrukcji zależne od spełnienia warunku,
- for each (*element*) do ... - wykonanie iteracyjne pewnej grupy instrukcji dla zdefiniowanego zbioru elementów (pojemniki, dokumenty, czynności),
- thread ... - zdefiniowanie wątku instrukcji; wątki mogą być wykonywane współbieżnie w definiowanym procesie,
- synchronize (*thread1, thread2, ..., threadN*) - synchronizacja wątków, będąca warunkiem przejścia do wykonania dalszych instrukcji; jeśli jeden z wątków nie zakończy się, to proces nie będzie mógł być kontynuowany.

Przykłady

Poniżej zaprezentowano przykłady opisu w języku PMDL dwóch czynności: recenzja oraz zatwierdzenie dokumentu. Obie czynności polegają na zmianie statusu dokumentu.

Czynność Recenzja

Czynność Recenzja ma następujące parametry:

- pojemnik, z którego zostanie pobrany dokument,
- deskryptor dokumentu, który ma zostać poddany recenzji (inspekcji).

Dokument zostaje pobrany z pojemnika, a następnie jego status zostaje zmieniony na *inspected* (lista dopuszczalnych wartości statusu dokumentu jest definiowana wcześniej). Poniżej podano definicję czynności *Recenzja* w języku PMDL.

```
define Status as set (draft, inspected, approved);
declare inspection as Activity;
inspection.create('Recenzja', {bucket as ProductContainer, docDescr as
ProdcutDescr})
{
 declare doc as Product;

 doc = bucket.get(docDescr);
 doc.changeStatus(inspected);
}
```

Poniżej podano przykład użycia czynności *Recenzja* w przykładowym procesie. Pewien dokument zostaje opublikowany w pojemniku zawierającym dokumenty specyfikacyjne. Następnie dokument ten poddany zostaje recenzji.

```
define Status as set (draft, inspected, approved);
declare fts as Product;
declare ftsDescr as ProdcutDescr;
```

```
declare specBucket as ProductContainer;

ftsDescr = specBucket.put(fts);
inspection.execute({specBucket, ftsDescr});
```

Czynność Zatwierdzenie

Czynność *Zatwierdzenie* ma następujące parametry:

- pojemnik, z którego zostanie pobrany dokument,
- deskryptor dokumentu, który ma zostać zatwierdzony.

Dokument zostaje pobrany z pojemnika, a następnie jego status zostaje zmieniony na *approved*, pod warunkiem jednak, że dokument został poddany wcześniej inspekcji. Nie może zatem zostać zatwierdzony dokument nierecenzowany. Poniżej podano definicję czynności *Recenzja* w języku PMDL.

```
define Status as set (draft, inspected, approved);
declare approvement as Activity;
approval.create('Zatwierdzenie', {bucket as ProductContainer, docDescr as
ProdcutDescr})
{
 declare doc as Product;

 doc = bucket.get(docDescr);
 if (doc.getStatus() is inspected) then doc.changeStatus(approved);
}
```

Kolejny przykład ilustruje użycie czynności *Zatwierdzenie* w przykładowym procesie. Dokument zostaje opublikowany w pojemniku zawierającym dokumenty specyfikacyjne. Następnie dokonywana jest recenzja dokumentu oraz jego zatwierdzenie.

```
define Status as set (draft, inspected, approved);
declare fts as Product;
declare ftsDescr as ProdcutDescr;
declare specBucket as ProductContainer;
ftsDescr = specBucket.put(fts);
inspection.execute({specBucket, ftsDescr});
approval.execute({specBucket, ftsDescr});
```

Odwzorowanie opisu procesu w języku PMDL w architekturę SoFar

Proces opisany za pomocą języka PMDL obejmuje wszystkie fazy funkcjonowania systemu *SoFar*, opisane w rozdziale 4 (por. rys. 3). Obejmuje on fazę odkrywania, zawiera bowiem również definicje węzłów oraz przypisanie fizyczne pojemników do węzłów sieci (osiąga się to przez przypisanie określonych wartości do atrybutów odpowiednich obiektów). Obejmuje również fazę zbierania informacji z uprzednio zdefiniowanych pojemników. W tej fazie główną rolę odegrają specjalizowani agenci

(nieożywieni uczestnicy procesu), działający na pojemnikach i na produktach. Następne fazy procesu bazują na informacjach zebranych w tej fazie. Wprawdzie faza organizowania nie występuje jawnie w opisie procesu (w istocie jest ona realizowana przez wytwórców i kierowników projektów i w aktualnej wersji nie podlega opisowi w języku PMDL), to jednak efekty tej fazy, w postaci dokumentów raportujących aktualny stan projektu, występują w opisie procesu i stają się przedmiotem kolejnej fazy - rozpowszechniania. Podczas tej fazy dokumenty są przekazywane dalej, na wyższe poziomy zarządzania, a przekazywanie to realizowane jest w ramach wątków wykonywanych przez nieożywionych (agentów) i ożywionych (ludzi) uczestników procesu.

Jasne jest, że do wdrożenia procesu *SoFar* niezbędny jest generator („kompilator" PMDL), który na podstawie opisu procesu zapisanego w języku PMDL utworzy stosowne elementy architektury *SoFar*, przede wszystkim zestaw pojemników oraz agentów, którzy będą nadzorowali i realizowali przepływy produktów pomiędzy pojemnikami. Te elementy architektury muszą być następnie umieszczone (osadzone) w intranecie organizacji i na żądanie menedżera zainicjowane. Taki generator jest aktualnie przedmiotem intensywnych prac.

Podsumowanie i dalsze prace

W niniejszym rozdziale zaprezentowano koncepcję zbierania informacji na temat procesu wytwórczego w organizacji wytwarzającej oprogramowanie w celu poprawy tego procesu, a ściślej - w celu usprawnienia zarządzania tym procesem. Rozwiązanie takie jest alternatywą dla typowej procedury dokumentowania projektu informatycznego w postaci repozytorium dokumentacji projektowej. Zaproponowane rozwiązanie pozwala na przyrostowe tworzenie systemu, poczynając od poziomu wytwórcy, poprzez poziom kierownika projektu, aż do poziomu menedżera organizacji. Najbliższe prace zostaną ukierunkowane na zastosowaniu języka PMDL do opisu dużych projektów informatycznych oraz - po zrealizowaniu opisanego w poprzednim punkcie generatora - wykonanie studium przypadku w jednej z większych firm informatycznych Wybrzeża.

Bibliografia

[BAY1999] M. E. Bays, *Software Release Methodology*, 1999, Prentice-Hall.

[BTS1999] A. Berson, H. Thearling i S. J. Smith, *Building Data Mining Applications for CRM*, 1999, McGraw Hill Professional Publishing.

[COHU1995] M. Cotterell i B. Hughes, *Software Project Management*, 1995, International Thomson Publishing Co..

[GOR2000] J. Górski, *Inżynieria oprogramowania w projekcie informatycznym*, 2000, MIKOM.

[HACK1999] R. D. Hackathorn, *Web Farming for the Data Warehouse*, 1999, San Francisco: Morgan Kaufmann Publishers.

[KIMB1996] R. Kimball, *The Data Warehouse Toolkit ? Practical Techniques for Building Dimensional Data Warehouses*, 1996, New York: John Wiley & Sons.

[MENA2001] J. Mena, *Web Mining for Profit ? E-business Optimization*, 2001, Butterworth-Heinemann.

[OMG2002] *Object Management Group (OMG)*, 2002, http://www.omg.org.

[RAT2002] *Rational ? The Software Development Company*, 2002, http://www.rational.com.

[SZEJ2002] S. Szejko, *Metody wytwarzania oprogramowania*, 2002, MIKOM, Warszawa.

[W3C2002] *World Wide Web Consortium*, 2002, http://www.w3.org.