PRACE NAUKOWE AKADEMII EKONOMICZNEJ WE WROCŁAWIU
Nr xxx 2004
Nowoczesne technologie informacyjne w zarządzaniu

Lech Madeyski, Marcin Kruczkiewicz

Politechnika Wrocławska,
Wydział Informatyki i Zarządzania, Wydziałowy Zakład Informatyki
e-mail: lech.madeyski@pwr.wroc.pl, kruq@e-informatyka.pl

Rozszerzalny obiektowy model projektu informatycznego

Streszczenie: Artykuł dotyczy problemów związanych z narzędziami wspomagającymi realizację małych i średnich projektów informatycznych. Zaprezentowano pomysł na rozszerzalny obiektowy model projektu informatycznego (ang. *eXtensible Project Object Model*) oraz zaproponowano otwartą i rozszerzalną architekturę realizacji systemu bazującą na XML. Specyfikacja wymagań wobec systemu była wynikiem realnych potrzeb firmy realizującej małe i średnie projekty informatyczne. W rezultacie powstało narzędzie o ciekawej, rozszerzalnej (za pomocą wtyczek) architekturze wspomagające realizacje projektów informatycznych.

1. Klasyczne sposoby wspomagania zarządzania projektem informatycznym

Każdy projekt informatyczny wymaga zespołu programistów, specyficznego dla projektu środowiska developerskiego IDE (ang. *Integrated Development Environment*) oraz wsparcia narzędzi wspomagających prowadzenie projektu. Zazwyczaj pojawia się potrzeba zgłaszania błędów, przechowywania kart wymagań, planowania zadań oraz gromadzenia informacji z nimi związanych, proponowania nowych cech i funkcjonalności, usprawniania komunikacji z klientem. Stosowanie wielu darmowych, ogólnodostępnych programów jest nieefektywne, a monolityczne systemy są drogie i mało elastyczne.

Praktyka pokazuje, że większość uczestników projektów rezygnuje ze stosowania narzędzi właśnie z powodów uciążliwości obsługi i strat czasu. Błędy zgłaszane są pocztą elektroniczną, lista zadań jest trzymana w notatniku, a ważne informacje są przechowywane w różnych miejscach w postaci luźnych,

pojedynczych dokumentów. Również częsta jest sytuacja, w której programista najpierw wykonuje całą zaplanowaną na pewien okres pracę, a następnie uzupełnia wymagane wpisy do narzędzi wspomagających projekt. W takim wypadku dane są wprowadzane w pośpiechu i niedbale. Często zdarza się, że pojedyncze zadania są kumulowane i wprowadzane jako jeden spójny wpis, co skutecznie utrudnia analizę postępów projektu i może prowadzić do mylnych wniosków.

W artykule przedstawiona zostanie próba rozwiązania, opisanego wyżej, poważnego problemu polegającego na niespójności informacji o projekcie. Rozwiązanie problemu polega na stworzeniu elastycznej, rozszerzalnej architektury bazującej na mechanizmie wtyczek (ang. *plugins*) oraz implementacji rdzenia systemu wraz z przykładowym rozszerzeniem. Funkcjonalność powinna być na tyle podstawowa, by nie zależeć od przyjętego modelu i metodyki wytwarzania aplikacji. Otwarta architektura pozwalająca w dowolny sposób poszerzać i dopracowywać narzędzie do własnych potrzeb jest istotna, gdyż podstawowa funkcjonalność z założenia nie jest wystarczająca do zaspokajania potrzeb różnych zespołów. Sposób poszerzania funkcjonalności aplikacji (z punktu widzenia osób chcących tworzyć wtyczki) powinien być prosty i czytelny. Autorzy postulują postrzeganie projektu jako obiekt, który posiada atrybuty tworzące spójną całość i na którym można wykonywać metody, a nie jako listę zadań, zbiór programistów czy koszt przedsięwzięcia.

2. Proponowane rozwiązanie

Zalet obiektowego postrzegania problemów nie sposób przecenić. Obiektowość na stałe wrosła w świat informatyki. Dlaczego zatem nie pomyśleć o projekcie informatycznym jak o obiekcie? Wszelkie informacje związane z projektem stają się jego atrybutami (oczywiście w dobrze przemyślanej strukturze hierarchicznej) a ponieważ są one przechowywane w jednym bycie – o wiele łatwiej jest je wzajemnie logicznie powiązać, przetwarzać i rozszerzać. Zdaniem autorów do takich celów najlepiej nadaje się język XML (McLaughlin, 2001). Dodatkowo język ten (w istocie metajęzyk i dlatego mówi się o aplikacjach XML) pozwala na wykorzystanie istniejących mechanizmów komunikacji i transportu danych, a dzięki transformacjom obiekt projektu może być dowolnie reprezentowany w różnych formach – czytelnych dla człowieka, lub zrozumiałych dla innych systemów. XML pozwala również na sprawniejszą integrację z przyszłymi wtyczkami, dzięki całej gamie gotowych parserów dostępnych dla każdego języka programowania i każdej platformy. Idąc dalej można zauważyć że wraz z językiem XML gotowym sprostać potrzebom reprezentowania obiektu projektu, do dyspozycji jest mechanizm zawężania dokumentów XML – XML Schema (Van Der Vlist, 2001), który można wykorzystać do określenia jakie dokumenty XML reprezentują poprawny obiekt projektu a jakie nie. W takiej

sytuacji XPOM można przyrównać do instancji klasy wyrażonej za pomocą dokumentu XML Schema. W schemacie można określić wszelkie ograniczenia, możliwe zbiory atrybutów, oraz restrykcje co do samej struktury obiektu.
Zmiany zachodzące w projekcie można porównać do wykonywania metod obiektu projektu przenoszących go z jednego stanu w drugi, co wiąże się z modyfikacją atrybutów. Idąc dalej tym tropem można powiedzieć, że dodanie wtyczki to odziedziczenie bazy i jej udoskonalenie. Wtyczka może poszerzyć listę atrybutów oraz dostarczyć nową funkcjonalność. Opisane wyżej rozwiązanie nazwano Rozszerzalnym Model Obiektu Projektu – XPOM (ang. *eXtensible Project Object Model*). Podobną koncepcje wykorzystano w projekcie Maven (*Apache Maven*) realizowanym pod egidą organizacji Apache

3. Realizacja

Zdecydowano się podjąć próbę realizacji wyżej przedstawionych założeń wyznając trzy podstawowe kroki:

- specyfikacja wymagań (w celu głębszego zrozumienia problemu, oraz pozyskania wymagań do bazowej funkcjonalności),
- implementacja bazowej wersji systemu,
- implementacja wtyczki poszerzającej funkcjonalność bazową.

Poniżej zostały omówione poszczególne etapy całego procesu.

Specyfikacja wymagań

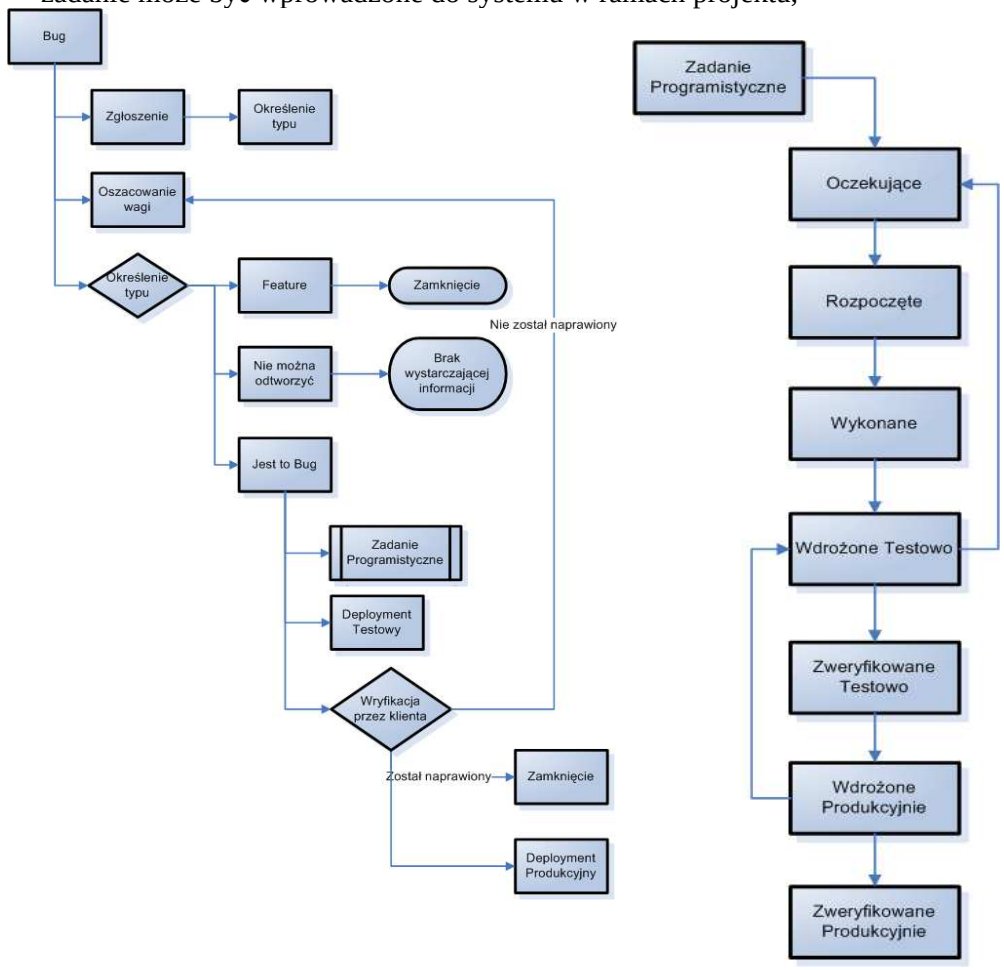
W celu zrozumienia jak funkcjonują małe i średnie przedsięwzięcia informatyczne, poddano analizie wrocławską firmę Power Media, której zespół dwunastu programistów realizuje biznesowe aplikacje internetowe. Specyfikacja była tworzona w czasie cyklicznych spotkań pracowników firmy. W spotkaniach brały udział trzy osoby wnoszące do specyfikacji różne sposoby postrzegania projektów:

- kierownik finansowo–organizacyjny odpowiedzialny za pozyskiwanie kontraktów,
- kierownik techniczny, nadzorujący realizacje projektów, biorący również udział w negocjacjach z klientami,
- programista.

W czasie wspólnych rozmów analizowane były poszczególne elementy związane z realizacją projektów, najczęściej występujące problemy, potrzeby klientów oraz zespołu projektowego. Specyfikacja rozpoczęła się od zidentyfikowania ’przepływów’ zachodzących w firmie. Przykładowe przepływy prezentuje rysunek 1. Jeden z przepływów ilustruje procedurę zgłaszania i obsługi błędów, drugi poszczególne etapy pracy nad zadaniem programistycznym. Gdy wszystkie przepływy zostały zidentyfikowane, doprecyzowywano najważniejsze

szczegóły specyfikacji. Przykładowo zadanie programistyczne zostało uszczegółowione w następujący sposób:

- zadanie programistyczne to podstawowa jednostka pracy,
- zadanie może być wprowadzone do systemu w ramach projektu,



Rysunek 1: Przykładowe przepływy: błąd oraz zadanie programistyczne.

- do zadania może zostać przypisany programista odpowiedzialny za jego realizacje,
- dowolny członek zespołu może przypisać odpowiedzialność za zadanie innemu programiście,
- spośród zadań do wykonania, programiści sami wybierają te, które im najbardziej odpowiadają (gdy ktoś jest niezdecydowany to zostaje mu to czego inni nie wybrali),

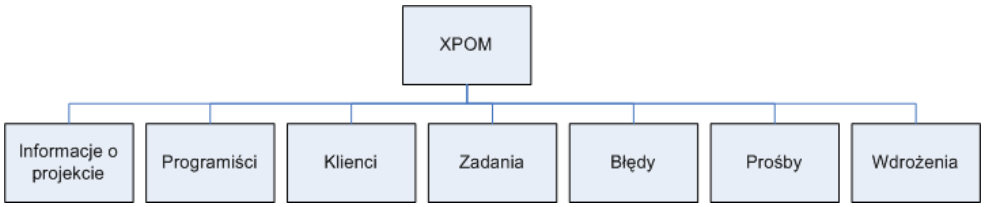
- po wprowadzeniu zadania przypisywane jest do niego oszacowanie czasu realizacji,
- zadanie może zostać przekazane innemu programiście przez osobę aktualnie za nie odpowiedzialną (w przypadku złego przyporządkowania, lub potrzeby wykorzystania kompetencji innej osoby),
- zadanie można podzielić na podzadania, by tworzyły hierarchie,
- do zadania można dołączać dowolna liczbę plików i notatek,
- zadania mogą być rożnych typów:
  - planowane (ang. *scheduled*) – zadanie wynikające z potrzeb projektu, jego realizacja jest niezbędna do wykonania otrzymanej od klienta specyfikacji,
  - błąd (ang. *bug*) – czynności które należy wykonać, aby wyeliminować pojedynczy błąd z systemu,
  - prośba (ang. *request*) – nowa cecha systemu, którą warto zaimplementować, wykryta już po zatwierdzeniu specyfikacji,
  - wyjaśnienie (ang. *question and answer*) – czas poświęcony na rozmowy dotyczące systemu (wyjaśnianie różnych kwestii najczęściej związanych z klientem),
- zadania mogą posiadać rożne statusy:
  - zgłoszone – pojawia się w systemie, ale nie rozpoczęto prac nad nim,
  - realizowane – w chwili obecnej trwają prace nad jego realizacją,
  - zrealizowane – zostało zrealizowane i czeka na wdrożenie,
  - wdrożone testowo – zostało wdrożone na serwerze testowym,
  - zweryfikowane testowo – zostało zweryfikowane przez klienta na serwerze testowym,
  - wdrożone produkcyjnie – zostało wdrożone na serwerze produkcyjnym,
  - zweryfikowane produkcyjnie – zostało zweryfikowane przez klienta na serwerze produkcyjnym,
- po wykonaniu zadania zostaje wprowadzony faktyczny czas jego trwania.

Tak przygotowana specyfikacja nie precyzuje dokładnie co i jak ma być zrobione, raczej pozwala nakreślić ramy i najważniejsze funkcjonalności systemu. Ponieważ celem tego etapu jest stworzenie bazowego modelu z najbardziej podstawową funkcjonalnością takie informacje powinny być jednak wystarczające.

Implementacja bazowej wersji systemu

Zrealizowanie funkcjonalności bazowej zakłada spełnienie pewnej grupy wymagań opisanych przez specyfikacje. Za przykład pozwalający prześledzić cały proces tworzenia systemu od specyfikacji poprzez implementację bazowej funkcjonalności po rozszerzenie (wtyczkę) posłuży nam najważniejszy element: Zadanie Programistyczne.

Ponieważ XPOM ma bazować na języku XML, do jego opisu wykorzystano mechanizm XML Schema, który:



Rysunek 2: Główne elementy hierarchicznej struktury drzewa XPOM.

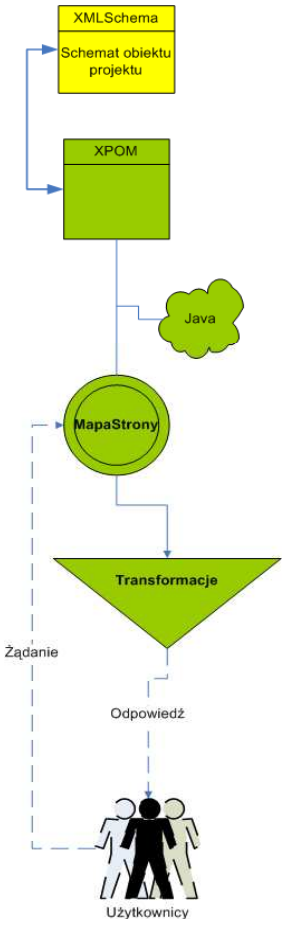
Ponieważ XPOM ma bazować na języku XML, do jego opisu wykorzystano mechanizm XML Schema, który:

- pozwala na automatyczna walidacje dokumentów XML przez większość istniejących parserów,
- pozwala na definiowanie własnych typów danych i na ich podstawie budowanie kolejnych, bardziej złożonych,
- posiada wiele predefiniowanych typów danych i zapewnia ich zrozumiałość bez względu na wykorzystaną platformie systemową,
- jest zrozumiała dla człowieka i posiada drzewiasta strukturę,
- na jej podstawie można stworzyć schemat relacyjnej bazy danych (istnieje grupa narzędzi pozwalających na wykonywanie takich mapowań automatycznie).

Jak już wspomniano, pierwszym zadaniem do wykonania po pozyskaniu specyfikacji było stworzenie na jej podstawie modelu danych XPOM. Element główny schematu XPOM przedstawia rysunek 2. Pokazuje on jak podzielono najważniejsze informacje o projekcie w podgrupy.

Na podstawie schematu XML Schema można tworzyć poprawne obiekty projektu, na dodatek zyskuje się mechanizm walidacji dla takowych reprezentacji. Powyżej przedstawiono tok postępowania od pozyskania specyfikacji do opracowania modelu danych i zapisania go w formie schematu XML Schema pozwalającego automatycznie sprawdzić poprawność dokumentu XPOM. Kolejnym krokiem jest dostarczenie logiki (tak jak obiekt jest bytem posiadającym pola oraz metody).

XPOM jest modelem projektu. Jak wspominano na początku artykułu, siła XPOM tkwi w jego niezależności od sprzętu, platformy systemowej oraz języka programowania oraz jego rozszerzalności.



Rysunek 3: Idea rozwiązania w kontekście Apache Cocoon

Implementacja logiki XPOM sprowadza się do operowania na drzewach XML. W praktyce logikę zaimplementowano w postaci klas Javy a aplikację osadzono na szkielecie publikacji Apache Cocoon (*The Apache Cocoon Project*, 2003). Idea proponowanego rozwiązania przedstawiona została przedstawiona na rysunku 3.

W prezentowanym rozwiązaniu modyfikowanie obiektu projektu odbywa na podstawie argumentów żądania klienta aplikacji internetowej, natomiast warstwa prezentacji bazuje na mechanizmach transformacji XSL (z drzewa obiektu projektu XPOM odpowiednie dane są transformowane do języka HTML).

Przykładową funkcjonalnością należącą do rozwiązania bazowego jest prezentowanie w postaci hierarchicznej informacji o zaplanowanych zadaniach w projekcie, wraz z informacją kto jest za nie odpowiedzialny, w jakim zadanie jest stanie oraz jakie estymacje są mu przypisane (rysunek 4).

Implementacja wtyczki poszerzającej funkcjonalność bazową

Funkcjonalność przykładowej wtyczki powinna być prosta (aby pokazać zasadę tworzenia bez komplikowania logiki), operować na nowych informacjach (a wiec pokazywać jak wtyczka ma dokładać dane do XPOM) oraz nie ingerować w bazową implementacje. Przykładowa wtyczka ma za zadanie umożliwić użytkownikowi dodawanie/usuwanie/modyfikację wag zadań. Takiej funkcjonalności nie posiada bazowy obiekt projektu. Informacji o wadze zadania także nie uwzględnia XML Schema. Wtyczka wykrywana jest automatycznie po starcie aplikacji. Nie zachodzi potrzeba jej rejestrowania ani dopisywania w plikach konfiguracyjnych. Dodanie wtyczki polega na skopiowaniu jej katalogu do podkontekstu wtyczek, oraz dodanie bibliotek wtyczki do bibliotek aplikacji. Wzorcowa wtyczka zajmująca się przechowywaniem informacji o wadze zadań trzyma dane w swoim katalogu w postaci pliku XML (jako zestaw par [ID zadania, waga zadania]). Gdy zachodzi potrzeba, wtyczka odczytuje obiekt projektu, następnie przetwarza swoje drzewo z informacjami o wagach zadań oraz łączy je w jedno spójne drzewo, gdzie każde zadanie oprócz danych pochodzących z XPOM posiada informacje o wadze. Tak połączone dane wtyczka przetwarza (np. dokonuje obliczenia średniej wagi dla zadań lub prezentuje listę zadań analogiczną do tej z bazowej implementacji, ale wzbogaconą o wagi – rysunek 4). W takiej sytuacji dołączone są informacje o wadze zadania. Warto podkreślić ze **oba** widoki są dostępne cały czas i równolegle – są od siebie **niezależne**. Wtyczka oczywiście musi zapewnić możliwość edycji swoich danych oraz sposób ich efektywnej prezentacji.

Rysunek 5 pokazuje wynik działania wtyczki – obliczanie dwóch średnich wag dla zadań: pierwsza waga liczona określa stosunek sumy wag do ilości zadań w projekcie, natomiast druga – stosunek sumy do ilości zadań posiadających przypisane wagi.

	Task	Importance	Time Estimated	Time Realized	Developer	Type	Status
1	Projekt bazy danych		23:00:00		Anzy	Planned	Waiting
5	Identyfikacja Encji		5:00:00		Kruq	Planned	Waiting
6	Identyfikacja zwiazkow		5:00:00		Kruq	Planned	Waiting
7	Normalizacja	8	3:00:00		Kruq	Planned	Waiting
8	Weryfikacja zgodnosci ze specyfikacja		5:00:00		Anzy	Planned	Waiting
9	Implementacja projektu w MsSQL Server		6:00:00		Kruq	Planned	Waiting
2	Implementacja interfesju	12	2004-04-20 2004-05-20		Kowpaw	Planned	Waiting
3	Implementacja logiki	2	2004-05-01 2004-06-17		Kruq	Planned	Waiting
10	Implementacja SecureRequestHandler		15:00:00		Kowpaw	Planned	Waiting
11	Mechanizm przechwytywania zadan		2:00:00		Kowpaw	Planned	Waiting
12	Analizator uprawnien	12	5:00:00		Kruq	Planned	Waiting
13	ret	13			Kruq	Planned	Waiting
4	Testy obciazeniowe		2004-07-15 2004-07-25		Kruq	Planned	Waiting

Rysunek 4: Lista zadań w projekcie widoczna poprzez wtyczkę

Ten prosty przykład pokazuje że wtyczka może wykorzystywać dane do przeprowadzania analiz (dane pochodzą z obiektu bazowego jak i z wtyczki).

Average importance for ALL tasks in this project is:

3.466666666666667

Average importance for MARKED tasks in this project is:

8.666666666666666

Rysunek 5: Dodatkowa funkcjonalność oferowana przez wtyczkę.

4. Wnioski i kierunki dalszych badań

Pokazano ze można zbudować narzędzie o architekturze wtyczkowej wspomagające prowadzenie projektów. Narzędzie o otwartej budowie, czerpiące z dobrych wzorców, wykorzystujące najnowsze technologie i łatwe w rozbudowie. Należy mieć na uwadze, że opisane prace są esencją większego przedsięwzięcia, natomiast w artykule zaprezentowano jedynie najważniejsze jego aspekty. W ramach prac:

1. **Zidentyfikowano problemy** związane z wykorzystaniem narzędzi wspomagających realizację projektów.
2. Przeanalizowano **przyczyny** powyższego stanu rzeczy.
3. Zaproponowano **nowy (**prosty, elastyczny i otwarty**) sposób** opisu projektu bazujący na obiektowym postrzeganiu projektu informatycznego.

4. Przeprowadzono **specyfikacje** wymagań systemu na rzeczywistej firmie realizującej projekty informatyczne i posiadającej zespół dwunastu programistów.
5. Na podstawie specyfikacji, **zbudowano klasę** bazowego obiektu projektu przy wykorzystaniu technologii XML Schema
6. Zbudowano przykładowy **obiekt projektu** (XPOM w języku XML) zgodny z klasa, opisujący przykładowy projekt.
7. **Zaimplementowano** fragment logiki obiektu projektu – związany z zadaniami, jako aplikacje internetowa na platformie J2EE
8. Zdefiniowano i zaimplementowano wzorcowa wtyczkę **poszerzającą** funkcjonalność systemu bez ingerencji w bazowa implementacje.
9. **Dowiedziono**, ze zakładane podejście jest realizowalne w praktyce i zdaniem autorów atrakcyjne.

Ponieważ całe przedsięwzięcie łączyło wiele dziedzin informatyki (miedzy innymi zarządzanie projektem, inżynieria wymagań, aplikacje internetowe, modularna architektura systemów, nowe technologie reprezentacji danych XML+XML Schema), niniejszy artykuł jest pracą pokazującą proces i sposób rozwiązania problemu a nie gotowym systemem. Przeprowadzenie pełnej specyfikacji wymagań, implementacja brakującej funkcjonalności bazowej oraz opracowanie zestawu wtyczek w znaczący sposób udoskonalających zaproponowana bazę wymagałaby o wiele większych nakładów pracy.

Warto budować systemy otwarte i rozszerzalne, oparte na mechanizmach przyjmowania wtyczek. Obecnie rynek jest ogromny, a co za tym idzie ogromna jest różnorodność pojawiających się potrzeb. Sensownym wydaje się również budowa systemów z ograniczoną funkcjonalnością bazową przygotowaną na przyjmowanie rozszerzeń tak, by każdy miał możliwość komponowania funkcjonalności. Takie podejście do budowy narządzi przynosi następujące korzyści:

- klient dostaje funkcjonalność taką jakiej potrzebuje,
- klient płaci tylko za funkcjonalność jakiej potrzebuje,
- zawsze istnieje możliwość zastosowania własnych modyfikacji.
- szeroki wachlarz wtyczek rozwiązujących takie same problemy tworzy konkurencje a co za tym idzie poprawia jakość produktu,
- o wiele mniejsze zasoby są wymagane do tworzenia wtyczek, niż budowania od podstaw narzędzi posiadających potrzebną funkcjonalność.

Niewątpliwą zaletą systemu jest jego otwartość. Budowa wtyczek dopasowujących funkcjonalność systemu jest prosta. Wiązanie danych, nadpisywanie standardowych metod oraz czerpanie z mocy przeróżnych języków implementacji daje praktycznie nieograniczone możliwości – od integracji z innymi systemami po narzędzia wspierające implementacje (np. skrypty raportowania i kompilacji systemu) do dokładania informacji o aspekcie finansowym projektu z możliwością ich analizowania.

Artykuł ilustruje ideę oraz proces pozwalający na rozwiązanie postawionego problemu. Istnieje możliwość kontynuacji prac w zakresie:

- doprecyzowania specyfikacji, a co za tym idzie modelu bazowego,
- dokończenia implementacji tak, by logika pokrywała wszystkie potrzeby zidentyfikowane w specyfikacji,
- przyjęcia pewnego kierunku ewolucji systemu oraz opracowania zestawu wtyczek pozwalających realizować założone funkcjonalności (np. poprzez wtyczki dopasowujące system do wspierania projektów realizowanych w wybranej metodyce),
- prace nad możliwością tworzenia wtyczek wykorzystujących już istniejące rozszerzenia (np. zdefiniowanie standardowych, obowiązkowych metod dla wszystkich wtyczek dzięki którym możliwe będzie udostępnianie danych obiektu bazowego wzbogaconego o dane kolejnych wtyczek),
- propagowanie idei, gromadzenie społeczności zainteresowanej tworzeniem wtyczek na potrzeby różnych projektów.

Wykorzystanie zaprezentowanej idei rozszerzalnej architektury zostało zilustrowane na przykładzie wspomagania zarządzania projektem. Potencjalny obszar zastosowań może być jednak znacznie szerszy.

Literatura

Tenison J. (2001) *XSLT and XPath.* M&T Books.

McLaughlin B. (2001) *Java i XML*. Helion.

Van Der Vlist E. (2001) *XML Schema*. O'Reilly

Apache Maven, http://maven.apache.org

XML Schema, http://www.w3.org/XML/Schema

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

eXtensible Project Object Model

Abstract: This paper concerns problems of small and medium IT projects supporting tools. This research shows the idea of XPOM (eXtensible Project Object Model) and open, extensible architecture based on XML. Requirements specification of the system based on real world needs from a company that realize small and medium-sized projects. As a result a tool that has flexible and open architecture and is ready to use extensions functionality supplied by plugins has been proposed.