# OPHELIA – zintegrowane środowisko wytwarzania oprogramowania*

Maciej Hapke, Andrzej Jaszkiewicz, Krzysztof Kowalczykiewicz,
Dawid Weiss, Piotr Zielniewicz
*Instytut Informatyki, Wydział Informatyki i Zarządzania, Politechnika Poznańska,*
*{hapke, jaszkiewicz,krzysztof.kowalczykiewicz, dawid.weiss,*
*piotr.zielniewicz}@cs.put.poznan.pl*

**Streszczenie**

*Artykuł ten jest próbą podsumowania projektu Ophelia, którego celem jest opracowanie środowiska integrującego istniejące narzędzia wytwarzania oprogramowania we wspólną platformę poszerzającą ich funkcjonalność. Prezentujemy podstawowe założenia projektu oraz niektóre z nowych koncepcji, które okazały się bardzo interesujące z punktu widzenia inżynierii oprogramowania, jak traceability czy też dostęp do artefaktów projektu za pomocą uogólnionych interfejsów.*

# 1. Wstęp

Współczesne metodyki wytwarzania oprogramowania posługują się wieloma artefaktami jak kod źródłowy, raporty o błędach, harmonogramy, testy czy dokumentacja. Narzędzia programistyczne wspomagają tworzenie i zarządzanie tymi artefaktami. Często każde z nich pochodzi od innego producenta – różnią się wiec one zarówno zakresem, w jakim pokrywają proces produkcji oprogramowania jak i oferowaną funkcjonalnością. Narzędzia te są też najczęściej wyspecjalizowane w pełnionej przez siebie funkcji; są dedykowane do zarządzania wymaganiami, do modelowania oprogramowania, do tworzenia dokumentacji lub innej jeszcze innego podzbioru funkcji. Rzadko kiedy jeden dostawca ma w swojej ofercie produkty pokrywające cały proces wytwarzania

* OPHELIA jest wspólnym projektem realizowanym przez sześciu partnerów ze środowisk akademickich i przemysłowych, współfinansowanym z funduszy piątego programu ramowego Unii Europejskiej. Numer grantu: IST-2000-28402/ IST-2000-28402D. http://www.opheliadev.org

oprogramowania, które na dodatek mogą się ze sobą komunikować i wymieniać dane, ułatwiając tym samym pracę nad projektem programistycznym.

Obecnie jednym z najbardziej palących problemów inżynierii oprogramowania jest utrzymywanie spójności między artefaktami projektu. Dla przykładu, pamiętanie o odzwierciedlaniu zmian w kodzie źródłowym w opisujących go diagramach UML jest koszmarem większości programistów, chyba że czynność ta jest wspierana przez narzędzia CASE wyposażone w możliwości reinżynierii oprogramowania (ang. *round-trip engineering*) oraz integrację ze środowiskiem programistycznym. Ten sam problem dotyczy oczywiście aktualizacji dokumentacji, testów, harmonogramów i innych elementów projektu. Jest to zresztą ogólna trudność w reagowaniu na zmiany i propagowanie tych zmian wewnątrz projektu tak, by utrzymać spójność jego elementów. Dodatkowy koszt wymagany przez skrupulatną analizę zależności i na przykład formalny proces propagacji zmian jest zazwyczaj większy niż potencjalne zyski. Dlatego też znaczny odsetek projektów informatycznych porzuca wszelką dokumentację i modele UML po przekroczeniu pewnego etapu skomplikowania. W ścisłej integracji narzędzi programistycznych widzimy możliwość częściowego zautomatyzowania większości pracochłonnych i nużących czynności związanych z propagacją zmian, a przez to przyczynienie się do zwiększenia jakości projektu. Niestety, jak to już wspomnieliśmy, większość osiągalnych na rynku narzędzi to produkty zamknięte, komercyjne, którym brakuje możliwości integracji z innymi (szczególnie pochodzącymi od konkurencyjnych firm).

Można wyróżnić dwa podejścia do rozwiązania powyższego problemu integracji narzędzi. Pierwsze wymaga wyprodukowania, lub powiązania ze sobą takiego zestawu narzędzi, który pozwoli na zbudowanie ujednoliconego, rozproszonego środowiska zarządzającego wszystkimi elementami projektu. Przez rozproszenie mamy tu na myśli zarówno geograficzne rozproszenie wykonawców projektu, jak i komponentów systemu (np. kod źródłowy modułu, który produkują jedni wykonawcy projektu znajduje się na ich lokalnym serwerze, kod źródłowy innego modułu znajduje się w fizycznie innej lokalizacji, a jednak logicznie widziane są one jako spójna całość). Istnieją producenci oprogramowania dostarczający zestawy o tym charakterze, jak np. [WWW2003a,c,d]. Komercyjne pakiety są jednak zwykle zbyt drogie, i przez to nieosiągalne, dla małych bądź średnich firm (z drugiej strony można postawić pytanie o celowość tak skomplikowanych rozwiązań w przypadku projektów o niskim nakładzie, tym problemem się nie zajmujemy). Pakiety o charakterze open-source, takie jak [WWW2003c] wymagają zaś przestawienia się na dedykowane dla danego pakietu narzędzia, co może być postrzegane jako trudność w firmach komercyjnych.

Innym podejściem jest próba wytworzenia szkieletowej architektury, która umożliwiałaby zestawienie *istniejących* narzędzi i utworzenie z nich logicznie jednej platformy służącej wytwarzaniu oprogramowania. Taka architektura musiałaby być nie tylko sumą narzędzi, które by integrowała, ale także przynosić pewną wartość dodaną, na przykład w formie ujednoliconego „widoku" elementów projektu (bez konieczności odwoływania się do narzędzi, które zostały użyte do ich utworzenia). Stąd już jedynie krok dzieliłby tę architekturę od możliwości definiowania relacji między elementami projektu na właśnie takim „abstrakcyjnym" poziomie, możliwości przechwytywania i generowania zdarzeń o

zmianach i tym podobnych. Projekt OPHELIA jest przykładem właśnie takiego podejścia do integracji narzędzi.

OPHELIA jest owocem pracy sześciu partnerów akademickich i przemysłowych. Wkład autorów niniejszego artykułu w projekcie ogniskował się na budowie modułu śledzenia zależności między elementami projektu (ang. *traceability*), jego integracji z modułem powiadamiania o zmianach (ang. *notification*), implementacji modułu zarządzania wymaganiami oraz na integracji z platformą komercyjnego programu do układania harmonogramów, Microsoft MS Project. Wyzwaniem było zarówno udostępnienie danych z MS Projecta dla celów OPHELII, jak i zdefiniowanie powiązań i zdarzeń na poziomie abstrakcyjnych elementów projektu, który ten projekt definiuje.

Pozostała część niniejszej pracy opisuje podstawowe założenia projektu OPHELIA oraz nasze doświadczenia uzyskane podczas pracy nad integracją narzędzi różnych typów. W rozdziale drugim przedstawiamy i porównujemy z OPHELIĄ istniejące pakiety integrujące narzędzia wytwarzania oprogramowania. Rozdział trzeci opisuje projekt OPHELIA w szczegółach, prezentując architekturę i komponenty systemu oraz strategie integracji istniejących narzędzi. W rozdziale czwartym opisujemy ciekawe perspektywy, jakie może wnieść użycie technologii OPHELIA. Rozdział piąty prezentuje implementację koncepcji OPHELII – system Orpheus. Rozdział szósty to prezentacja dotychczasowych dokonań w projekcie i zarys planów na przyszłość.

# 2. Inne pakiety integrujące narzędzia programistyczne

Na rynku istnieje garść rozwiązań o podobnych do OPHELII założeniach. Tabela 1 przedstawia sumaryczne porównanie tych produktów.

Tabela 1. Porównanie cech pakietów integrujących narzędzia do wytwarzania oprogramowania (* - ograniczone wsparcie)

| Cecha | OPHELIA (Orpheus) | Rational Suite | Eclipse | Sourceforge Enterprise |
|---|---|---|---|---|
| Otwartość środowiska (publiczne interfejsy) | Tak | Nie* | Tak | Nie |
| Open source | Tak (z wyjątkiem modułu harmonogramowania) | Nie | Tak (z wieloma komercyjnymi pluginami) | Tak/Nie |
| Praca grup rozproszonych | Tak | Tak | Nie | Tak |
| Zarządzanie wymaganiami | Tak | Tak | Nie | Tak * |
| Zarządzanie harmonogramem | Tak | Nie | Tak * | Tak |
| Zarządzanie błędami | Tak | Tak | Tak * | Tak |
| Repozytorium plików | Tak | Tak | Tak | Tak |
| Śledzenie powiązań między elementami projektu | Tak | Tak * | Nie | Tak * |
| Wsparcie modelowania CASE | Tak | Tak | Tak * | Nie |
| Metryki | Tak | Tak | Nie | Tak * |
| Środowisko wytwarzania oprogramowania IDE | Tak * | Tak* | Tak | Nie |
| Moduł testowania oprogramowania | Nie | Tak | Tak * | Nie |
| Definicja formalnego procesu wytwarzania oprogramowania | Tak * | Tak | Nie | Tak |

## 2.1. Rational Suite

Rational Suite [WWW2003a] jest zbiorem narzędzi obejmującym uznane narzędzia CASE oraz zarządzania wymaganiami. Pakiet ten zapewnia dobrą integrację poszczególnych narzędzi oraz wspomaga pracę w środowisku rozproszonym. Jest to jednak stosunkowo wysoko wycenione narzędzie komercyjne trudno z tego powodu dostępne szczególnie dla małych i średnich firm.

## 2.2. Eclipse

Pakiet Eclipse [WWW2003b] był pierwotnie utworzony w laboratoriach IBM, został następnie przekazany do dalszego rozwoju jako produkt open-source i cieszy się wielkim powodzeniem, szczególnie w środowisku programistów technologii Java. Jest to rozszerzalne środowisko IDE (ang. *Integrated Development Environment*), które właściwie stanowi jedynie pewną otoczkę programową dla osadzanych w nim narzędzi. Narzędzia te, zwane wtyczkami (ang. *plug-ins*), różnicują się znacznie w swoim stopniu skomplikowania od bardzo prostych (widok pliku logu, konsoli), aż do bardzo skomplikowanych (modelowanie CASE).

Wydaje się, iż pakiet Eclipse, mimo iż na razie bardziej zorientowany na pracę lokalną, może stać się docelowo integratorem, a przynajmniej interfejsem graficznym, dla rozwiązań ujednolicających narzędzia w środowiskach rozproszonych.

## 2.3. Sourceforge Enterprise

Strona WWW firmy Sourceforge [WWW2003c] stała się ostatnio jedną z najczęściej odwiedzanych witryn środowisk open-source. Z usług Sourceforge korzysta ponad sześćdziesiąt tysięcy projektów (stan z maja 2003 roku).

Istnieje wersja oprogramowania do pracy grupowej o nazwie Sourceforge Enterprise [WWW2003d], która jest dedykowana dla instytucji komercyjnych. Składa się ona z internetowego systemu poszerzonego o dedykowane narzędzia do zarządzania projektem, śledzenia wymagań i inne moduły.

# 3. OPHELIA - środowisko integrujące narzędzia i elementy projektu

OPHELIA jest międzynarodowym projektem sponsorowanym z funduszy piątego programu Unii Europejskiej. Projekt rozpoczął się we wrześniu 2001 roku i ma przewidywaną datę zakończenia na wrzesień 2003 roku. Jego celem jest zaprojektowanie i następnie wdrożenie w praktyce technologii pozwalającej na zunifikowanie dostępu do narzędzi wytwarzania oprogramowania, a przez to zintegrowanie ich w jedną logiczną platformę.

Główną ideą OPHELII jest oparcie się na definicji ogólnych interfejsów programowych do różnych *typów* narzędzi. Te ogólne interfejsy stanowią punkt dostępu do elementów projektu i do samego narzędzia. Przykładowo, narzędzie zarządzania wymaganiami posiada ściśle zdefiniowany (ale ogólny) interfejs pozwalający na dostęp do każdego wymagania, pobranie jego właściwości (nazwy, atrybutów), zapisywanie się na powiadomienia w przypadku wprowadzenia do danego wymagania zmian. Aby udostępnić dane narzędzie zarządzania wymaganiami dla OPHELII, musi ono udostępnić ten właśnie interfejs (wewnętrznie, bądź przy pomocy pośredniego adaptera). Wszystkie

specyfikacje interfejsów OPHELII są zdefiniowane w języku IDL dla technologii CORBA.

Aktualnie w OPHELII jest zdefiniowanych wiele interfejsów do najpopularniejszych typów narzędzi wytwarzania oprogramowania. Taki „pakiet" dla danego narzędzia nazywa się MIS (ang. *Module Interface Specification*). W skład podstawowej dystrybucji OPHELII wchodzą następujące MISy:

- Usługi jądra platformy OPHELIA (ang. *Kernel services*)
- Moduł wymagań (ang. *Requirements module*)
- Moduł modelowania oprogramowania (ang. *Modeling module*)
- Moduł zarządzania harmonogramem projektu (ang. *Project management module*)
- Moduł metryk oprogramowania (ang. *Metrics module*)
- Moduł zarządzania błędami (ang. *Bug tracking module*)
- Moduł repozytorium plików (ang. *Repository module*)
- Moduł zarządzania wiedzą – zunifikowanego dostępu do dokumentacji (ang. *Knowledge management module)*
- Moduł zarządzania powiązaniami elementów projektu (ang. *Traceability Module*)
- Moduł powiadamiania o zmianach i komunikatów (ang. *Notifications module*)

W zależności od dostępnego oprogramowania i pożądanej konfiguracji, można więc zbudować wiele konfiguracji platformy OPHELIA. Ponieważ interfejsy są wyspecyfikowane ogólnie, więc konkretna implementacja (narzędzie) nie gra większej roli dla samej platformy. Oczywiście cicho zakładamy, iż narzędzia już eksponują stosowne implementacje interfejsów OPHELII. Jeśli tak nie jest, co na razie przynajmniej jest regułą, należy je do OPHELII dostosować. Można wyróżnić trzy typy takiego dostosowania, w zależności od specyfiki narzędzia, co przedstawia tabela 2.

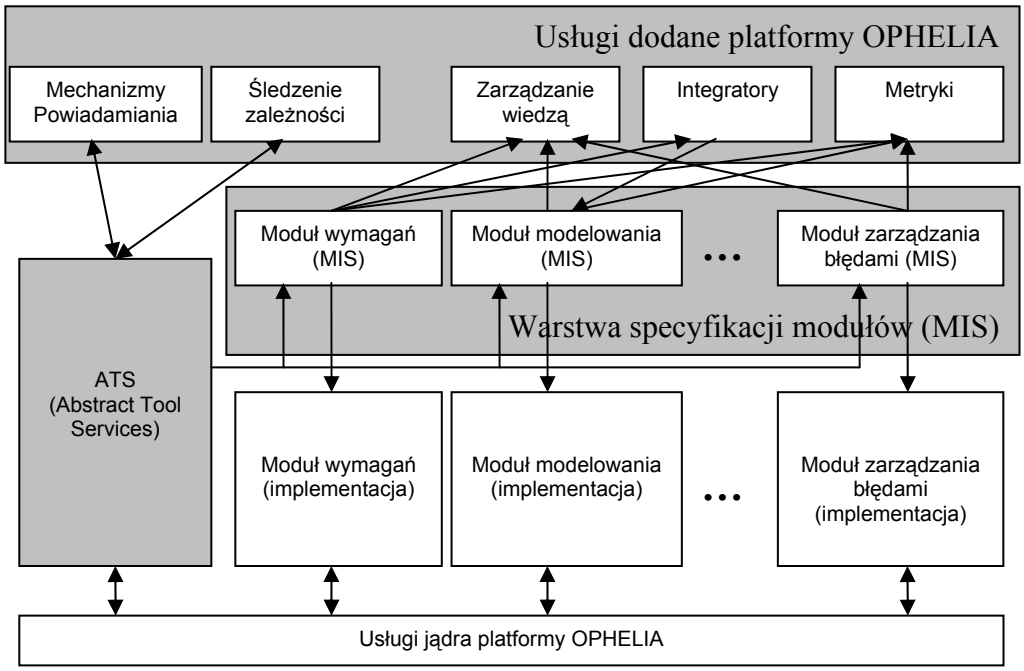
Tabela 2. Kroki wymagane by zintegrować
narzędzia programistyczne różnych typów z platformą OPHELIA

| **Typ narzędzia** | **Kroki potrzebne do integracji z OPHELIĄ** |
|---|---|
| Narzędzia lokalne (ang. *desktop*) | Należy napisać komponent serwera, który implementuje stosowny MIS.<br>Do samego narzędzia należy dopisać wtyczkę komunikującą się z serwerem. |
| Narzędzia klient-serwer | Należy napisać rozszerzenie komponentu serwera o implementację stosownego interfejsu MIS. |
| Narzędzia internetowe (ang. *thin client*) | Należy napisać adapter lub moduł serwera implementujący stosowny interfejs MIS. |

Jak widać z tabeli 2, aby współpracować z OPHELIA potrzebny jest do danego narzędzia komponent serwera, który udostępnia pozostałym komponentom platformy elementy projektu, którymi dany program się zajmuje. Jest to oczywiście dodatkowy wysiłek.

Konsorcjum OPHELIA ma jednak nadzieję, iż korzyści płynące z włączenia danego narzędzia do platformy staną się skutecznym magnesem, który skłoni producentów i środowiska open-source do jego podjęcia.

Owym magnesem mogą stać się dodatkowe serwisy i programy, które działają już na warstwie „abstrakcyjnych", uogólnionych interfejsów OPHELII. Można więc wydzielić mechanizmy bezpieczeństwa, administracji projektami i użytkownikami, ujednolicone adresowanie obiektów, mechanizm zdarzeń i ich propagacji, powiązań elementów projektu między sobą czy też wreszcie powiadamiania o zmianach dla użytkowników. Architektura OPHELII jest zaprezentowana na rysunku 1.



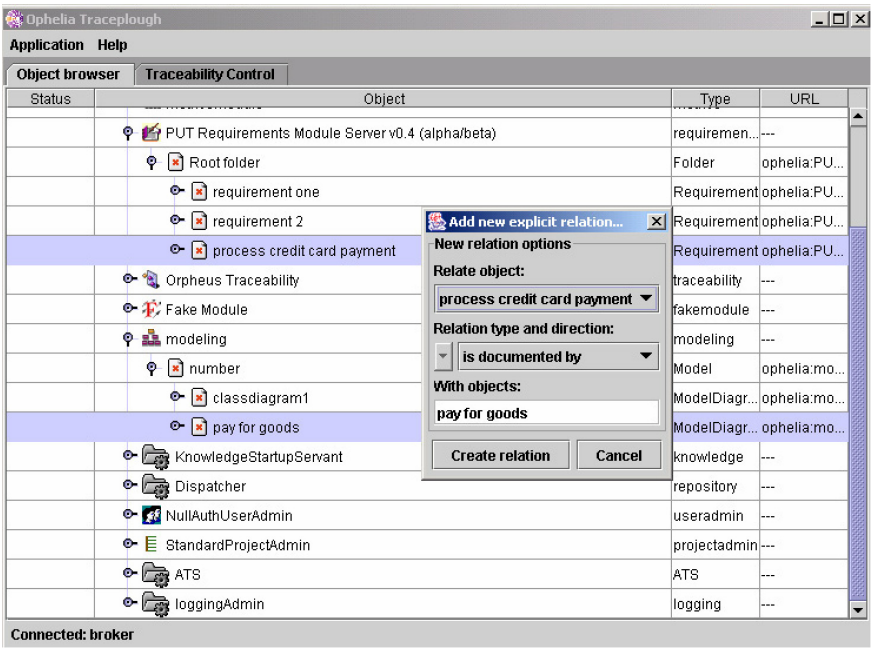

Rysunek 1. Architektura platformy OPHELIA

Podsumowując architekturę platformy OPHELIA, składa się ona z zestawu interfejsów dla określonych typów narzędzi programistycznych oraz z pewnych usług stanowiących wartość dodaną. OPHELIA stanowi więc odpowiednik szyny danych (ang. *data bus*) w architekturze komputerowej, gdzie odbywa się wymiana informacji między komponentami. Umożliwia ona składanie narzędzi programistycznych w jedną logiczną całość, którą wzbogacają dodatkowe usługi.

# 4. Dodatkowe usługi platformy OPHELIA

Uogólniona metoda dostępu do elementów projektu za pomocą MISów pozwala nam na zdefiniowanie zupełnie nowych typów usług, które trudno byłoby wprowadzić w życie

bez OPHELII [KOW2002]. Usługi te bowiem nie opierają się na znajomości konkretnych narzędzi, a jedynie na interfejsach MIS i funkcjach jądra OPHELII, z których czerpią wszelkie potrzebne im dane. Najprostszym przykładem jest tutaj najzwyklejsza przeglądarka (zob. rys. 2), która pokazuje *wszystkie* elementy składowe projektu, bez znaczenia jest bowiem to, jaki program (i gdzie został użyty do ich stworzenia). W pozostałej części tego rozdziału omawiamy inne ciekawe naszym zdaniem usługi dostępne w OPHELII.




Rysunek 2. Uogólniony widok elementów projektu oraz dodawanie zależności między elementami.

## 4.1. Narzędzia migracji danych (integratory)

Narzędzia migracji danych, nazywane przez nas integratorami, są typowym przykładem wykorzystania interfejsów OPHELII. Tego typu aplikacje są w stanie wydobyć informacje o elementach projektu jednego typu, przetworzyć je automatycznie lub z pomocą użytkownika oraz zapisać do innego modułu (narzędzia). Ułatwia to i skraca proces migracji danych z jednego typu narzędzia do drugiego, jednocześnie minimalizując wysiłek użytkownika wymagany przy tego typu operacji.

W OPHELII obecnie istnieje integrator pozwalający na półautomatyczną konwersję między diagramami klas i przypadków użycia (ang. *use-case diagrams*) a harmonogramem projektu. Podobnie można wyobrazić sobie aplikacje łączące moduł wymagań z modułem modelowania, kodu z dokumentacją i tym podobnie. Oczywiście funkcjonalność integratorów opartych jedynie o migrację danych jest ograniczona, ponieważ nasilają one jedynie efekt redundancji informacji w projekcie. Mamy nadzieję,

iż w zależności od typów modułów integracja ta może być jednak obustronna (synchronizacja) i w znacznym stopniu zautomatyzowana. Można wyobrazić sobie usługi działające w tle, które będą w stanie doprowadzać dane z różnych modułów do stanu spójności. Stopień zaawansowania aplikacji integrującej determinuje tutaj na ile owa synchronizacja będzie zautomatyzowana (OPHELIA jedynie dostarcza mechanizmy do komunikacji międzymodułowej).

## 4.2. Warstwa usług ogólnych: Abstract Tool Services

ATS jest w zasadzie nie tyle usługą, co pośrednikiem między jądrem OPHELII a modułami i usługami końcowymi. Jest on jednak na tyle ważny, iż wymaga osobnego opisu.

Interfejsy MIS definiują publiczny i uogólniony sposób dostępu do danego typu narzędzia internetowego. Eksponują one obiekty, które dany moduł udostępnia, zdarzenia, jakie generuje i inne informacje. Jednak dla każdego typu modułu ten zestaw cech jest inny. Za pomocą ATSu moduły i programy zewnętrzne są w stanie manipulować obiektami OPHELII, zapisywać się na powiadomienia o nich, dodawać relacje i tym podobne. Na pewien sposób ATS stanowi uogólnienie i tak już ogólnych MISów.

## 4.3. Usługa powiązań między obiektami: traceability

Usługa tworzenia i analizy powiązań między obiektami, nazywana z angielska traceability, jest jedną z podstawowych, które mają na celu ułatwienie pracy z projektem. Traceability umożliwia na definiowanie typów relacji i wiązanie elementów obiektu między sobą za pomocą tych typów (zob. rys. 2). Nadal nieważna pozostaje fizyczna lokalizacja obiektu i narzędzie użyte do jego utworzenia (dostęp do obiektu gwarantuje OPHELIA). Przykładowo można w prosty sposób powiązać wymaganie z modelem, model z kodem źródłowym, kod źródłowy z testami i tak dalej. Graf relacji może być później analizowany i przeglądany w zależności od potrzeb. Jako dodatkową funkcję, OPHELIA oferuje również możliwość propagacji zmian danego elementu projektu do obiektów od niego zależnych (właśnie korzystając z relacji utworzonych wcześniej).

W projekcie informatycznym mogą już istnieć pewne relacje, na przykład zaszyte w konkretnych narzędziach lub utworzone przez opisywane wyżej integratory. Mapowanie tych relacji do traceability przedstawia tabela 3.

Tabela 3. Typy relacji między elementami projektu i ich odpowiedniki w traceability

| **Typ relacji** | **Opis** |
|---|---|
| Wewnątrzmodułowe | Relacje znajdujące się w konkretnych narzędziach, np. zależnościowe między plikami w kodzie źródłowym. Takie relacje należy ręcznie wprowadzić do modułu traceability. Na razie automatyzacja w tym zakresie jest słaba. |
| Zewnętrzne dodawane niejawnie | Tworzone przez same komponenty OPHELII, na przykład integratory. |
| Zewnętrzne dodawane jawne | Tworzone jawnie przez użytkowników Ophelii w aplikacji klienckiej modułu traceability. |

Należy podkreślić, iż graf relacji między obiektami jest bardzo ciekawym obiektem z punktu widzenia inżynierii oprogramowania. Pozwala on na badanie wpływu potencjalnej zmiany jeszcze przed jej wprowadzeniem do systemu, czy też stopnia skomplikowania i zależności międzymodułowych uwzględniając pełen kontekst wszystkich elementów projektu [KOW2002, DEW2003].

## 4.4. Usługa powiadomień o zmianach

Dzięki usłudze powiadomień, użytkownik może otrzymywać informacje o zdarzeniach, jakie mają miejsce w projekcie. W szczególności chodzi tu o informowanie o zmianach. Użytkownicy i managerowie, zakładając, iż wcześniej wyrazili chęć bycia informowanym o zmianach danego obiektu, będą informowani za każdym razem, gdy zmiana następuje na samym obiekcie, jak i na obiektach, do których prowadzą określone relacje wyspecyfikowane za pomocą modułu śledzenia zależności. Przykładowo, w momencie wprowadzenia zmiany do wymagania, informowani są użytkownicy, którzy są przypisani do opieki nad stosownymi diagramami, które to wymaganie modelują. Podobnie w momencie zmiany diagramów, aktualizacji powinien również ulec kod oraz zapewne dokumentacja i zbiory testujące. Propagacja powiadomień następuje w OPHELII automatycznie.

Narzuca się pytanie, czy w przypadku skomplikowanych i gęstych grafów zależności między elementami projektu nie zajdzie sytuacja ignorowania powiadomień przez użytkowników. Wydaje się, że ta sytuacja implikuje głębszy problem z architekturą projektu. Cena użyteczności powiadomień nie jest nam jednak na razie znana i jej zbadanie jest ważnym celem pilotowych wdrożeń platformy.

## 4.5. Globalne zarządzanie wiedzą

Moduł globalnego (w sensie projektu) zarządzania wiedzą zajmuje się zarówno aktualizacją dokumentacji, jak i umożliwia poszukiwanie elementów projektu różnych

typów na zasadzie wyszukiwarki. Moduł „nasłuchuje" również na zdarzenia dziejące się w systemie i odpowiednio uaktualnia dokumentację.

Szczególnie interesujące z technicznego punktu widzenia jest to, że sam moduł zarządzania wiedzą jedynie wysyła zapytanie do ATSu i później składa z informacji uzyskanych z modułów całą dokumentację. W efekcie więc to moduł danego narzędzia generuje takie informacje na temat danego obiektu, by były one jak najbardziej reprezentatywne.

## 4.6. Metryki

Na podobnej, co moduł zarządzania wiedzą, zasadzie działa moduł metryk projektu. Zdefiniowane dla danych elementów metryki są automatycznie przeliczane w momencie zachodzenia zmian na obiektach, dla których są one liczone. Pozwala to na śledzenie zmian metryk w czasie (ang. *event-driven metrics history*). Obiecujące w tym kontekście wydaje się szczególnie utworzenie automatycznych usług powiadamiania o niepokojących zmianach lub przekroczeniu wartości progowych metryk. Taki system „wczesnego ostrzegania" byłby z pewnością nieoceniony przy analizie jakości i zarządzaniu projektem.

# 5. Orpheus – implementacja architektury OPHELIA

Jednym z kluczowych zadań projektu OPHELIA było zrealizowane przedstawionych koncepcji w formie prawdziwego produktu, który mógłby być wykorzystany do realnego wytwarzania oprogramowania. Produkt ten, nazwany Orpheus, jest darmową implementacją napisaną w języku Java, licencjonowaną na zasadzie open-source. Implementacja obejmuje zarówno usługi jądra OPHELII, jak i adaptery do wielu typów narzędzi. Istniejące produkty, jak ArgoUML czy Bugzilla zostały rozszerzone o serwer implementujący odpowiedni interfejs MIS. W paru przypadkach (jak np. moduł zarządzania wymaganiami) napisane zostały własne produkty i ich połączenia z OPHELIĄ. Jedynym przypadkiem użycia narzędzia komercyjnego jest Microsoft Project, do którego dopisano moduł serwera.

Tabela 4 prezentuje zestawienie typów narzędzi i produktów użytych jako ich konkretne instancje w produkcie Orpheus.

Tabela 4. Moduły programowe składające się na produkt Orpheus

| Typ narzędzia | Instancja w produkcie Orpheus |
|---|---|
| Usługi jądra platformy OPHELIA | Orpheus Broker, implementacja własna, open-source |
| Moduł wymagań | DRES, implementacja własna, open-source |
| Moduł modelowania oprogramowania | ArgoUML, produkt open-source<br>Moduł serwera, implementacja własna, open-source |
| Moduł zarządzania harmonogramem projektu | Microsoft Project, produkt komercyjny<br>Moduł serwera, implementacja własna, open-source |
| Moduł zarządzania błędami | Bugzilla, produkt open-source<br>Moduł serwera, implementacja własna, open-source |
| Moduł metryk oprogramowania | The Metrix, implementacja własna, open-source |
| Moduł repozytorium plików | CVS, produkt open-source<br>Adapter udostępniający dane dla OPHELII, implementacja własna, open-source |
| Moduł zarządzania wiedzą | Knowledge Module, implementacja własna, open-source |
| Moduł zarządzania powiązaniami elementów projektu | Traceability Module, Traceplough, implementacja własna, open-source |
| Moduł powiadamiania o zmianach i komunikatów | Notification Module, Traceplough, implementacja własna, open-source |
| Usługi migracji danych | Moduł modelowania → Moduł zarządzania harmonogramem, implementacja własna, open-source |
| Portal internetowy | Portlet dla Apache Jetspeed, implementacja własna |

# 6. Podsumowanie działalności projektu

Projekt OPHELIA stanowi łączny owoc badań, stworzonego oprogramowania i ewaluacji wyników. Głównym zamierzeniem było dostarczenie specyfikacji platformy integrującej narzędzia wytwarzania oprogramowania oraz stworzenie prototypu, który by tę specyfikację implementował. Początkowo rozważano utworzenie scentralizowanego repozytorium elementów projektu, w którym zaszyta byłaby wszelka logika konieczna do realizacji innych funkcji platformy. Ostatecznie jednak zdecydowano się na model rozproszony, gdzie każdy typ narzędzia jest reprezentowany przez oddzielny moduł, zaś

ich współpraca i unifikacja widoku przechowywanych przez nie danych dostarcza nową jakość.

Pierwsze problemy pojawiły się już na tym etapie, gdy przy definiowaniu interfejsów MIS należało zdecydować się na określenie granularności, z jaką eksponowane są elementy projektu poszczególnych typów (na przykład: całe diagramy UML, czy pojedyncze ich części składowe).

Niedługo po wykrystalizowaniu się koncepcji architektury powstały pomysły na jej wykorzystanie w celach dużo śmielszych niż oryginalnie zakładane w planie projektu. Okazało się, że OPHELIA umożliwia wprowadzenie nowych usług jak śledzenie powiązań między obiektami lub metryki sterowane zdarzeniami, które przed momentem zaistnienia OPHELII były trudne, lub niemożliwe do realizacji. W dalszej perspektywie postanowiono udostępnić użytkownikom portal internetowy, który miał stać się scentralizowanym punktem dostępu do narzędzi i usług platformy.

Główne zadania programistyczne zostały przydzielone na lato i jesień roku 2002, pod koniec którego powstała wersja alfa produktu Orpheus. Wersja beta, oficjalnie opublikowana latem roku 2003 jest przedmiotem wdrożeń u partnerów przemysłowych i źródłem pierwszych doświadczeń z użycia w prawdziwych projektach. Wyniki nie są jeszcze znane autorom w momencie pisania tej pracy.

# 7. Zarys planów na przyszłość

W momencie zakończenia oficjalnego czasu trwania projektu OPHELIA konsorcjum pragnie kontynuować popularyzację koncepcji i technologii utworzonych w czasie jego trwania. Planuje się utworzenie usług konsultingowych dla firm pragnących wdrożyć rozwiązania OPHELII we własne procesy produkcji oprogramowania. Planuje się również wsparcie dla szerszego wachlarza narzędzi, w tym również komercyjnych.

Akademiccy członkowie projektu w szczególności cenią sobie nowe perspektywy i możliwości analizy środowiska wytwarzania oprogramowania. Potencjalne kierunki badawcze to między innymi:

- Specyfikacje pozostałych (innych) typów narzędzi nie uwzględnionych obecnie w OPHELII (np. analiza ryzyka, workflow, testowanie).
- Inteligentne monitorowanie procesu wytwarzania oprogramowania w oparciu o metryki generowane w odpowiedzi na zdarzenia oraz analizę danych historycznych.
- Włączenie formalnej definicji procesu obiegu dokumentów i artefaktów programowych do specyfikacji platformy.
- Ściślejsza integracja ze środowiskami IDE jak Eclipse oraz analiza zysków i kosztów wprowadzenia OPHELII do procesu wytwarzania oprogramowania.
- Dodanie nowych technologii podstawowych do jądra w technologii CORBA (np. Web Services).

# 8. Podsumowanie

OPHELIA jest przedstawicielem nowego podejścia do integracji narzędzi programistycznych. Opiera się ono na budowie warstwy niezależnych od samych narzędzi serwisów, oraz interfejsów, do których przyłączane są konkretne narzędzia w zależności od potrzeb danej organizacji. Do głównych zalet tego rozwiązania należą otwartość, niski koszt wykorzystania już istniejących usług, wsparcie dla pracy rozproszonej użytkowników a zarazem udostępnienie scentralizowanego widoku wszystkich elementów projektu. Niewątpliwie wielce obiecujące są również nowe usługi jak śledzenie powiązań, reakcja na zdarzenia, metryki z historią czy powiadamianie użytkowników o zmianach. Jesteśmy zdania, iż na bazie uogólnionych interfejsów do narzędzi programistycznych istnieje jeszcze wiele możliwych do zrealizowania koncepcji usług usprawniających pracę nad projektami informatycznymi.

W momencie publikacji tego artykułu wersja beta produktu Orpheus przechodzi fazę ewaluacji u użytkowników przemysłowych. Oczekujemy, iż przyniosą one odpowiedzi na wiele pytań dotyczących zarówno używalności samej koncepcji scentralizowanej architektury, jak i poszczególnych usług o nią opartych.

Do pewnego stopnia OPHELIA może być postrzegana jako rywal dla produktów komercyjnych wspomnianych w rozdziale 2. Wydaje się nam jednak, że w duchu open-source, jej istnienie może jedynie wspomóc inne produkty i zapoczątkować ruch integracyjny wśród producentów oprogramowania.

# Bibliografia

[BOL2003] Boldyreff C., Dewar R., et al., *Environments to Support Collaborative Software Engineering*. 2nd Workshop on Cooperative Supports for Distributed Software Engineering Processes, Benevento, Italy, March 25, 2003.

[DEW2003] Dewar R., Smith M., et al., *The OPHELIA Traceability Layer*. 2nd Workshop on Cooperative Supports for Distributed Software Engineering Processes, Benevento, Italy, March 25, 2003.

[HAP1999] Hapke M., Jaszkiewicz A., Kominek P., Integrated tools for project scheduling under uncertainty (in Polish). Proceedings of 1st National Conference on Software Engineering, Kazimierz Dolny 11□13.10.1999, Informatyka Stosowana S4/99, Politechnika Lubelska, 65□76.

[HAP2000] Hapke M., Kominek P., Jaszkiewicz A., Slowinski R., Integrated tools for software project scheduling under uncertainty. In: P. Brucker, S. Heitmann, J. Hurink, S. Knust (Eds.) *Proc. 7th Int. Workhop on Project Management and Scheduling PMS'2000*, Osnabrueck, Germany, April 17-19, 2000, pp.149-151.

[HAP2001] Hapke M., Jaszkiewicz A., Perani S.: *OPHELIA - Open platform and methodologies for development tools integration in a distributed environment*. Proceedings of 3rd National Conference on Software Engineering, Otwock/Warsaw, p. 189-198.

[KOW2002] Kowalczykiewicz K., Weiss D., Traceability: Taming uncontrolled change in software development. Proceedings of 4th National Conference on Software Engineering, Tarnowo Podgórne, Poland, 2002.

[WWW2003a] http://www.rational.com/products/entstudio/index.jsp

[WWW2003b] http://www.eclipse.org

[WWW2003c] http://www.sourceforge.net

[WWW2003d] http://www.vasoftware.com/products