Metodyka Select Perspective

Artur Kasprzyk
POTIS Software Development Tools
artur.kasprzyk@potis.com.pl

Streszczenie

Komponentowość stała się ostatnio bardzo modnym pojęciem, propagowanym praktycznie przez wszystkich twórców narzędzi wykorzystywanych w trakcie produkcji oprogramowania. Jednakże, obserwacja rynku twórców oprogramowania wykazuje, iż niewiele firm stosuje w swojej codziennej pracy spójne podejście prowadzące do wytwarzania, zarządzania oraz wykorzystania komponentów na skalę korporacyjną. Jednym z głównych powodów takiego stanu rzeczy jest brak wsparcia dla takiego stylu pracy ze strony „pierwszoplanowych" dostawców technologii. Odpowiedzią na to zapotrzebowanie jest pojawienie się kilka lat temu nowego wcielenia metodyki Select Perspective, która jest obecnie jednym z bardzo niewielu procesów produkcyjnych, kładących nacisk na komponentowość rozwiązania już na pierwszych etapach cyklu życia systemu. Podejście takie ma za zadanie zmniejszenie całkowitego kosztu wytwarzania systemów oraz umożliwienie jasnego rozdziału odpowiedzialności pomiędzy zespołami tworzącymi części składowe rozwiązania oraz zespołami odpowiedzialnymi za dostarczenie całego systemu, zgodnie z wymaganiami sponsorów przedsięwzięcia. Celem niniejszego rozdziału jest zaprezentowanie istoty metodyki Select Perspective oraz przedstawienie wsparcia, jakie oferują nowoczesne narzędzia CASE zespołom tworzącym oprogramowanie.

1. Wstęp

Współczesne metodyki w coraz większym zakresie adoptują obiektowe podejście do wytwarzania oprogramowania. Wraz z pojawieniem się pierwszych wersji języka Unified Modeling Language nastąpił wyraźny wzrost zainteresowania stosowaniem uporządkowanych obiektowych procesów produkcyjnych przez producentów oprogramowania. Tym samym podejście obiektowe trafiło „pod strzechy".

Kilka lat stosowania technologii obiektowej dowiodło, iż sama zmiana paradygmatu nie stała się katalizatorem oczekiwanego drastycznego wzrostu ponownego wykorzystania istniejących rozwiązań czy też wyraźnego podziału prac pomiędzy dostawców podzespołów i dostawców ostatecznych rozwiązań. W dalszym ciągu, pomimo zmiany podejścia do wytwarzania systemów informatycznych, model kaskadowy (w różnych wcieleniach) jest powszechnie stosowany, nie przystając już do realiów obecnie prowa-

dzonych przedsięwzięć. Alternatywą dla rozbudowanych metodyk stały się tak zwane zwinne metodyki (ang. *agile methodologies*), których twórcy wyraźnie kontestują monolityczne, rozbudowane cykle produkcyjne, takie jak np. Rational Unified Process, oferując w zamian koncentrowanie się na podstawowej czynności, jaką jest programowanie i wspierając się jedynie sporadycznie modelowaniem oraz kładąc silny nacisk na stałą współpracę ze specjalistami dziedzinowymi.

Rozwój metodyki Select Perspective ewoluował nie tyle w kierunku ograniczania poszczególnych faz projektu, ile na wypracowaniu podejścia, które z jednej strony będzie zapewniało wystarczający poziom faz koncepcyjnych (modelowania procesów biznesowych, analizy, projektu), skracając jednakże do minimum czas pojawienia się produktu finalnego u klienta końcowego. Metodą na osiągniecie celu, jest zdaniem twórców metodyki, stymulacja współbieżnego wytwarzania ortogonalnych składowych aplikacji, koncentrowanie się na usługach zamiast na konkretnych klasach, minimalizacji pośrednich produktów procesu oraz wyraźny podział na dostawców oraz odbiorców komponentów.

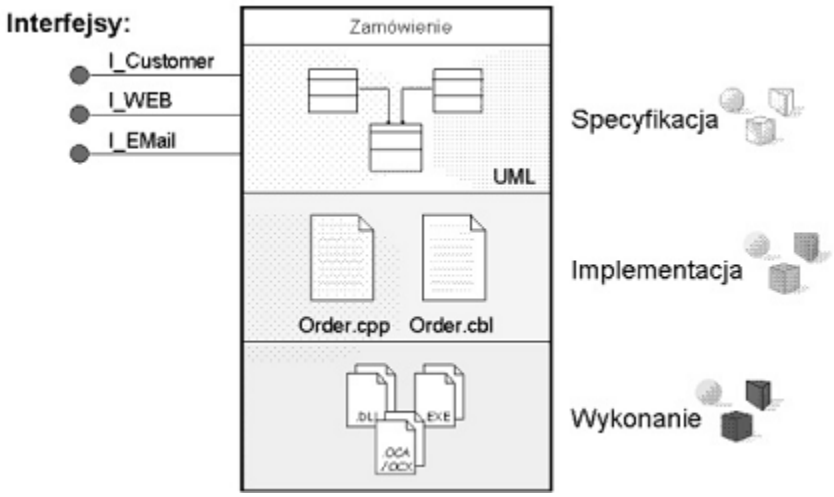
2. Komponenty

Obecna literatura jest bogata w różnorodne dyskusje dotyczące definicji komponentu. Metodyka Select Perspective na własne potrzeby specyfikuje następujące właściwości komponentu:

- Komponent jest elementem osadzanym w środowisku uruchomieniowym,
- Gotowe, całościowe rozwiązania składają się z komponentów,
- Komponenty komunikują się wzajemnie ze sobą, wykorzystując do tego celu ustandaryzowane protokoły wymiany informacji,
- Komponenty posiadają specyfikację świadczonych przez nie usług zdefiniowanych w postaci interfejsów.

Odwołując się do komponentowości, należy wspomnieć o poziomach abstrakcji, na jakich można postrzegać komponent (rysunek 1). Pierwszym z nich jest poziom specyfikacji komponentu (nazywany często fasadą). Można go interpretować jako deklarację usług, które komponent powinien realizować lub realizuje oraz specyfikacje usług, które współpracujący komponent powinien dostarczyć, aby można było nawiązać współpracę. Specyfikacja komponentu powinna być całkowicie wolna od jakichkolwiek aspektów technologicznych.

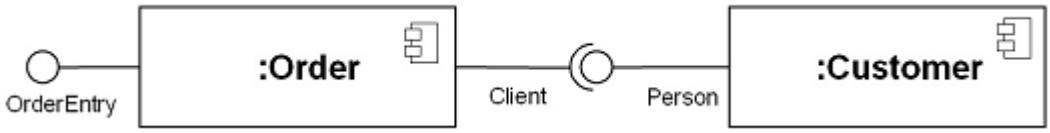
Implementację specyfikacji w danym języku programowania należy rozumieć jako kolejny poziom abstrakcji. Dla danej specyfikacji może istnieć wiele implementacji komponentu. Przykładem może być np. budowa dwóch wersji aplikacji rozproszonej przeznaczonej do realizacji przy wykorzystaniu standardów: CORBA (dla środowiska UNIX) i .NET Remoting (dla środowiska Windows). Każda z aplikacji będzie wymagała osobnej implementacji np. w językach C++ oraz C#.



Rysunek 1. Różne poziomy abstrakcji komponentu

Kolejną postacią komponentu jest jego reprezentacja w postaci plików wykonywalnych, czyli takich, które mogą być osadzone w docelowym środowisku uruchomieniowym.

Czwartą, ostatnią reprezentacją komponentu, jest już gotowy, skonfigurowany element, osadzony w środowisku uruchomieniowym. Osadzenie komponentu najczęściej wymaga powiązania go z innymi elementami środowiska poprzez zestaw interfejsów i portów, zdefiniowanych w komponentach. Przykład definicji konfiguracji (w notacji UML 2.0) przedstawia rysunek 2.



Rysunek 2. Współpraca komponentów (UML 2.0)

Podejście komponentowe do budowy systemów informatycznych nie ogranicza się jedynie do fazy definiowania komponentu. Komponentowość oznacza także budowę odpowiedniej infrastruktury umożliwiającej współdziałanie komponentów oraz testowanie komponentu zarówno jako samodzielnego bytu jak i składowej konkretnego rozwiązania. Tak więc komponentowość znajduje swoje odzwierciedlenie już na etapie analizy systemu, w trakcie, której precyzowane są oczekiwania wobec składowych systemu na podstawie wymagań organizacji opisanych modelem procesów biznesowych.

3. Korzyści, jakich oczekuje się od technologii komponentowej

Od bardzo wielu lat próbuje się wskazywać na pożądane podobieństwa (wykazując „niepożądane" różnice) pomiędzy inżynierią oprogramowania i innymi działami inżynierii. Symptomatyczne jest już posługiwanie się analogiami z dziedziny budownictwa, przy tłumaczeniu niektórych zasad inżynierii oprogramowania.

Wraz z pojawieniem się obiektowego podejścia do budowy systemów, głośno zrobiło się o wielokrotnym stosowaniu raz wytworzonych elementów (ang. *reusability*). Elementami wielokrotnie wykorzystywanymi miały być obiekty, będące reprezentantami klas. W praktyce okazało się, że poziom wielokrotnego wykorzystania obiektów (szczególnie wchodzących w skład warstwy logiki aplikacji) nie jest duży.

Sytuację tę ma znacznie poprawić technologia komponentowa. Elementem przewagi nad technologią obiektową - w jej „czystej postaci" - jest wyraźne oddzielenie warstwy specyfikacji komponentu od warstw realizacji. Dzięki temu, zasady składania aplikacji są określane na najwyższym (pierwszym) poziomie abstrakcji, natomiast obecnie dostępne rozwiązania technologiczne pozwalają na realizację opisanej współpracy bez względu na ostateczną realizacje komponentu (z ograniczeniem na wspólny protokół komunikacji).

3.1. Korzyści ekonomiczne i biznesowe

W czasach globalnego kryzysu firmy szukają rozwiązań pozwalających na bardziej efektywne i skuteczne wytwarzanie dóbr. Informatyka nie jest w tym względzie wyjątkiem – wydaje się, że bezpowrotnie minął już czas rozciągliwych budżetów przedsięwzięć informatycznych, przez co problem utrzymania inwestycji w planowanym budżecie stał się jednym z podstawowych zadań szefa projektu.

Technologia komponentowa ma przynieść wymierne ekonomiczne korzyści poprzez znaczne zwiększenie ponownego wykorzystania istniejących rozwiązań. Katalizatorem dla tego typu działań z jednej strony jest dojrzalsza technologia, z drugiej natomiast zauważalny wzrost zainteresowania producentów oprogramowania wdrażaniem sprawdzonych procesów wytwórczych, opartych o podejście komponentowe. O ile pierwszy argument wydaje się wartym zaprezentowania, o tyle drugi z wymienionych wydaje się na tyle zdroworozsądkowy, że przez domniemanie można byłoby założyć, że jest aktualny. I tutaj zdaje się dochodzimy do jednej z największych bolączek przemysłu informatycznego – pomimo piętrzących się od lat problemów, w dalszym ciągu firmy informatyczne pracują metodami „harcerskimi", broniąc się przed „formalizmami", jakie niesie wdrażanie procesów wytwórczych. Powracając do sedna sprawy, technologia obiektowa zdaje się wyznaczać nowy kierunek w produkcji oprogramowania, przenoszący ciężar pracy z mozolnego wytwarzania większości składowych systemów na jego składanie z jak największej liczby gotowych „prefabrykatów". Od podstaw powinien być docelowo wytwarzany jedynie ten kod aplikacji, który jest wymagany do sprzężenia prefabrykatów w jedną funkcjonująca sprawnie całość.

Opisane podejście, prócz zmiany stosunków pracy w zespołach (pojawiają się całkiem nowe role i obowiązki) wpływa w znaczący sposób na cały rynek wytwórców oprogramowania. Skoro systemy mają być wytwarzane z gotowych elementów, powinien powstać rynek tychże. Specyfika produkcji oprogramowania wymaga powstania rynku (w pewnym sensie, infrastruktury informatycznej), na którym informacje o oferowanych przez produkt usługach (czytaj, zaimplementowanej specyfikacji) powinny być dostępne *on-line*, z każdego stanowiska pracy, wymagającego takiej informacji. Powsta-

nie rynku komponentów w dużym stopniu wprowadzi do inżynierii oprogramowania normalne, zdrowe zasady rynkowe: możliwość wyboru spośród wielu dostępnych rozwiązań konkurencyjnych, powstanie katalogu półproduktów, itp.

Docelowo, przeniesienie ciężaru pracy z wytwarzania na składanie ma zmniejszyć całkowity koszt realizacji przedsięwzięć informatycznych oraz wpłynąć na zmniejszenie czasu ich realizacji (kluczowe zagadnienie w szybko zmieniających się zasadach prowadzenia działalności gospodarczej).

3.2. Korzyści techniczne

Z technicznego punktu widzenia komponentowość wprowadza kilka nowych elementów do procesu wytwórczego. Przede wszystkim, już na etapie pierwszych prac nad systemem (określanych często mianem analizy) należy położyć nacisk na wyróżnienie specyfikacji usług, które będą mogły w kolejnych fazach projektu być zrealizowane z użyciem prefabrykatów. Abstrahowanie od takiego widoku systemu spowoduje, iż niezwykle trudnym stanie się wkomponowanie istniejących już produktów w cały system.

Innym istotnym aspektem technicznym jest wyraźny podział ról na osoby tworzące specyfikacje komponentów, oraz zespoły je realizujące. Podział ról wyraźnie wyznacza granice odpowiedzialności, którą jest zgodność interfejsów (usług komponentów). Aspekty technologiczne (takie jak język implementacji, przyjęte rozwiązania projektowe) zamykają się w ramach jednego zespołu. Podział ról daje szanse na współbieżne wytwarzanie komponentów potrzebnych do realizacji przedsięwzięcia, skracając tym samym czas jego powstawania.

Technologia komponentowa powoduje także niewielką zmianę zasad testowania aplikacji: testy mogą być realizowane na rzecz prefabrykatów (komponentów), podsystemów (ang. *subsystem*) a następnie gotowych rozwiązań. Oznacza to (mając na uwadze wcześniej opisany podział ról w projekcie), że wartościowe testy integralnych części aplikacji mogą być prowadzone niezależnie (co oznacza między innymi, współbieżnie) od siebie.

Z technicznego punktu widzenia, zarządzanie zmianami powinno uwzględniać analizę wpływu na oferowane obecnie usługi komponentu oraz utrzymanie zgodności interfejsów (usług) komponentów przez cały cykl ich życia. Zasada ta jest podstawowym kryterium możliwości stosowania przyrostowego usprawniania komponentów.

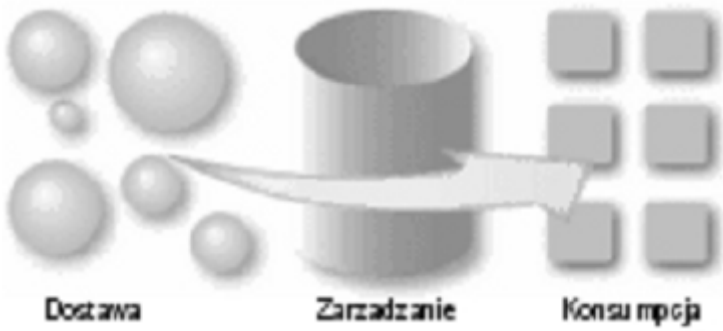
4. Metodyka Select Perspective™ - model Supply – Manage –Consume (SMaC)

Większość współczesnych modeli funkcjonowania organizacji opiera się na pojęciu łańcucha dostaw. Bazując na dostępnych na rynku półproduktach organizacja wytwarza określony produkt, który może stać się elementem kolejnego łańcucha dostaw, bądź też zostanie dostarczony finalnemu konsumentowi jako gotowe dobro.

W chwili obecnej większość z dostępnych metodyk wytwarzania oprogramowania zdaje się nie zauważać tej prostej i skutecznej zasady działania. Procesy produkcyjne in-

żynierii oprogramowania w dalszym ciągu są oparte o jeden monolityczny ciąg działań, promujący pracę od podstaw i abstrahujący od rozwijającego się rynku gotowych półproduktów. W efekcie tego, pomysły na wykorzystanie elementów już dostępnych najczęściej są efektem prac świadomych programistów (czasami projektantów); nie wynikają one natomiast z przemyślanej strategii determinowanej przyjętymi zasadami pracy.

Metodyka Select Perspective w obecnej fazie rozwoju (jej obecność na rynku datuje się od roku 1994) w bezpośredni sposób nawiązuje do idei łańcucha dostaw. Cały proces produkcyjny opiera się na wyróżnieniu trzech podstawowych rodzajów działalności, których wzajemne współistnienie prowadzi do budowy systemów informatycznych zgodnych z wymaganiami klientów oraz dostarczanych w jak najkrótszym czasie (por. rysunek 3). Procesem bezpośrednio odpowiedzialnym za dostarczenia gotowego produktu końcowego jest proces dostarczenia rozwiązania (ang. *consume*). Rozwiązanie jest składane z półproduktów, które mogą być wytworzone na podstawie specyfikacji określonej przez uczestników procesu dostarczania oprogramowania, bądź też wskutek analizy – pozyskane z rynku półproduktów już dostępnych. Za wytworzenie komponentów zgodnie z określoną specyfikacją odpowiedzialny jest proces dostawy półproduktów (ang. *supply*) – rysunek 4.

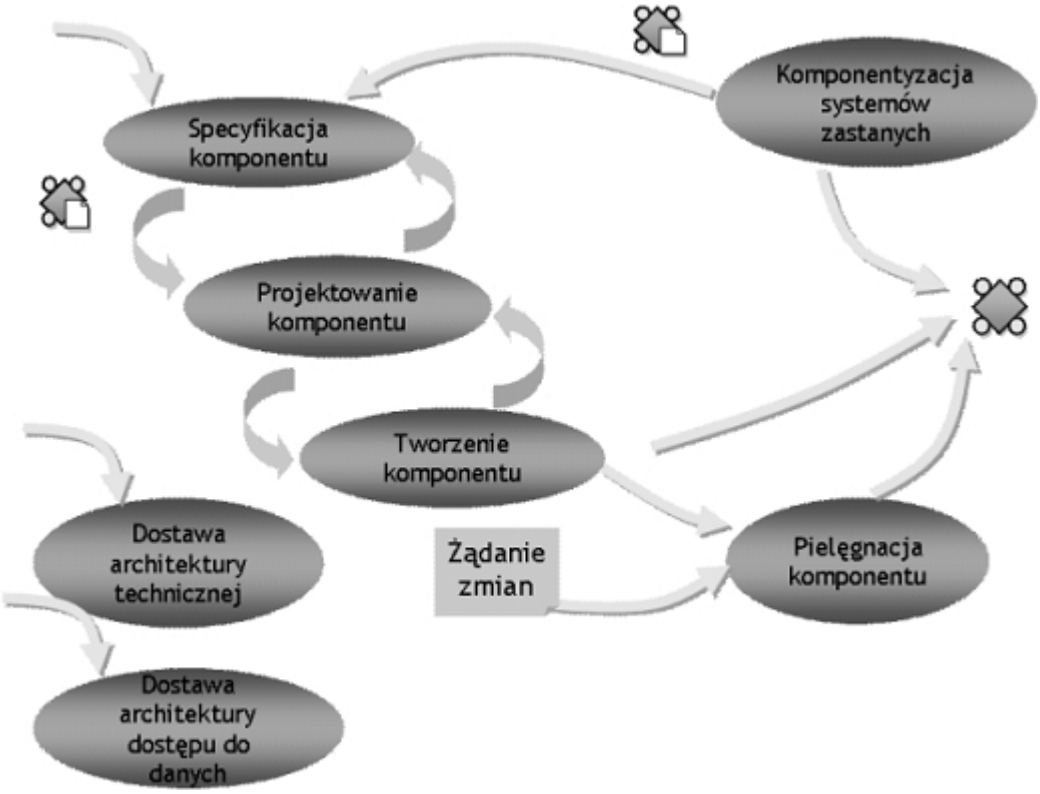


Rysunek 3. Istota podejścio Supply - Manage - Consume (SMaC)

Źródłem danych dla procesu dostarczania półproduktów mogą być bądź specyfikacje określone w procesie tworzenia rozwiązania (produktu finalnego), bądź też specyfikacje wynikające z komponentyzacji systemów zastanych. Drugie z wymienionych źródeł wyraźnie wskazuje na fakt, iż metodyka uwzględnia nie tylko budowę systemów nowych, ale także integrację z istniejącymi już produktami, które należy przystosować do pracy w nowym środowisku systemów zintegrowanych.

Określona początkowo specyfikacja usług podlega negocjacjom w trakcie jej realizacji przez zespół dostarczający rozwiązanie. Negocjacje mogą być na przykład wynikiem dostosowywania rozwiązania do specyfiki środowiska, efektem obsługi zmiany wymagań bądź też próby dostosowania istniejącego już półproduktu do nowych zastosowań.

Podstawowym rezultatem prac omawianego procesu jest gotowy do użycia, przetestowany zarówno pod kątem zgodności z wymaganiami użytkownika jak i możliwości osadzenia w zdefiniowanej architekturze technicznej komponent.

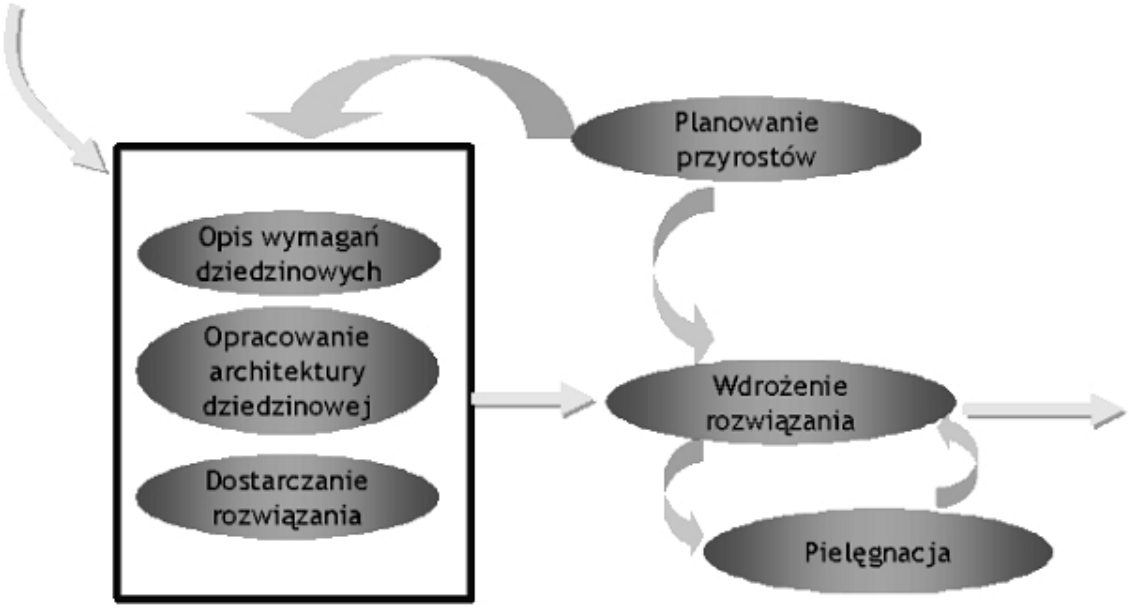


Rysunek 4. Proces tworzenia półproduktów

Proces dostawy komponentów, prócz omówionych już zadań, jest także odpowiedzialny za zdefiniowanie oraz dostarczenie odpowiedniej architektury środowiska uruchomieniowego, która będzie umożliwiała wykorzystanie budowanych oraz planowanych do zastosowania komponentów. Prócz infrastruktury dającej możliwość wzajemnej komunikacji komponentów, specjalnej rozwadze poddawany jest aspekt rozwiązania realizujący składowanie elementów rozwiązania w bazie danych.

Zapewnienie trwałości elementom rozwiązania jest problemem tym bardziej złożonym, im większy jest rozdźwięk pomiędzy pojęciami wykorzystywanymi do opisu rzeczywistości oraz jej implementacji w konkretnym środowisku a pojęciami specyficznymi dla bazy danych. Wydaje się, że w chwili obecnej najpopularniejszym środowiskiem wytwórczo - uruchomieniowym jest obiektowy język implementacji oraz relacyjna baza danych. Pogodzenie tych dwóch środowisk, jakże różnych pod względem semantycznym, spędza sen z powiek wielu projektantom i programistom - z punktu widzenia metodyki Select Perspective, - członkom zespołu odpowiedzialnego za dostawę komponentów, jako że oni to właśnie są dokładnie zaznajomieni ze specyfiką składowych ostatecznego rozwiązania. Jednym z możliwych (i najbardziej efektywnych z punktu widzenia kosztów) rozwiązań jest zastosowanie dostępnych na rynku narzędzi klasy „O-R mapping", których zadaniem jest automatyzacja zapisu obiektów w bazach relacyjnych (np. POTIS Object-Relational Toolkit dla środowiska Microsoft .NET, TopLink dla środowiska Java, Bold dla środowiska Delphi, itd.). Alternatywą dla zastosowania komercyjnych rozwiązań jest budowa własnego mechanizmu zapewniania trwałości. Jest to jednakże podejście sprzeczne z jedną z podstawowych zasad metodyki Select Perspecti-

ve, która mówi, że należy w jak największym stopniu wykorzystywać istniejące już rozwiązania. Jeśli nie można dopasować żadnego z nich, wówczas należy rozpocząć proces budowy elementu środowiska od podstaw (ang. *reuse before you buy, buy before you build*).



Rysunek 5. Proces konsumpcji komponentów

Podstawowym zadaniem projektu informatycznego jest wytworzenie oprogramowania zgodnego z oczekiwaniami użytkowników. W metodyce Select Perspective rozwiązanie takie powstaje w efekcie realizacji procesu konsumpcji komponentów – rysunek 5.

Proces konsumpcji komponentów obejmuje swoim zakresem zarówno wytworzenie produktu, jak i późniejsze wdrożenie oraz pielęgnację. Elementy procesu prowadzące do wytworzenia gotowego produktu zostały podzielone na trzy podstawowe etapy, które mogą być (bardzo często – powinny) rozpoczęte równocześnie. Dla przykładu, modelowanie przypadków użycia, prowadzone w ramach **Opisu wymagań dziedzinowych**, na pewnym etapie zatwierdzania wymaga stworzenia prototypu interfejsu użytkownika. Ten z kolei, jest tworzony w ramach realizacji **Dostarczania rozwiązania**.

Opis wymagań dziedzinowych ma za zadanie zebranie informacji o sposobie funkcjonowania organizacji, dla której tworzony jest system informatyczny, zebrania wymagań biznesowych, użytkownika (prezentowanych w postaci modelu przypadków użycia), funkcjonalnych oraz niefunkcjonalnych. Integralną częścią procesu jest definicja skryptów testowych, które będą wykorzystywane do sprawdzania zgodności wytworzonej aplikacji z wyspecyfikowanymi wymaganiami użytkownika. Efektem realizowanych w ramach tego etapu prac jest powstanie produktów pośrednich, które będą wykorzystywane w trakcie dalszych prac nad systemem:

- *model dziedzinowy*, na który składają się:
  - *model procesów biznesowych,*
  - *model przypadków użycia,*

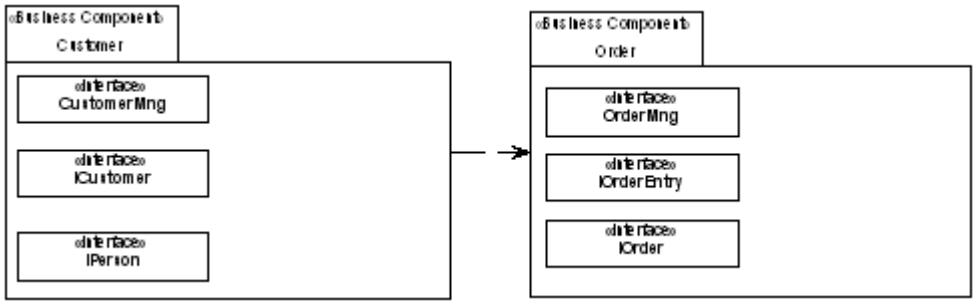
  - *katalog reguł biznesowych,*
  - *zbiór wymagań,*
  - *ograniczenia,*
- *plan testów zgodności aplikacji z oczekiwaniami użytkownika*.

Po zakończeniu prac związanych z opisem wymagań dziedzinowych, zespół dostarczający rozwiązanie posiada wystarczającą ilość informacji potrzebnych do uzasadnienia celowości realizacji projektu. Elementami, które powinny być rozpatrzone przed podjęciem dalszych prac są:

- *ryzyko niepowodzenia*. Znajomość zakresu przedsięwzięcia pozwala na identyfikacje tych czynników, które mogą negatywnie wpłynąć na realizowane przedsięwzięcie. Zidentyfikowane czynniki niepowodzenia powinny być poddane wnikliwej analizie, celem sporządzenia planu minimalizacji wpływu czynnika na całość prac.
- *wartość projektu*. Wstępne szacunki odnośnie kosztów prac oraz zysków, jakie przyniesie organizacji wdrożenie systemu, pozwalają na określenie bilansu potencjalnych korzyści.
- *plan projektu*. Znajomość zakresu przedsięwzięcia pozwala na utworzenie pierwszych wiarygodnych harmonogramów działań.
- *decyzje odnośnie zakresu*. Informacje zebrane w trakcie realizacji etapu pozwalają na podjęcie decyzji związanych z różnego rodzaju wariantami i opcjami, o których istnienie najprawdopodobniej sponsorzy i decydenci nie byli świadomi przed rozpoczęciem prac.

Opracowanie architektury dziedzinowej ma na celu identyfikację komponentów dziedzinowych oraz sprecyzowanie usług od nich oczekiwanych. W efekcie tych działań powstaje model koncepcyjny rozwiązania, wykorzystujący pojęcia zrozumiałe dla analityków, projektantów oraz programistów. Prace nad architekturą dziedzinową w naturalny sposób wpasowują się w iteracyjny cykl pracy. W zależności od etapu prac, kolejno specyfikowane są podstawowe komponenty, które będą składały się na rozwiązanie, następnie definiowane są ich odpowiedzialności w systemie oraz wzajemne zależności. Ustalenie wstępnych odpowiedzialności jest warunkiem koniecznym do analizy usług, jakie komponent powinien oferować. Dość często wyróżnienie usług staje się przyczynkiem do modyfikacji wcześniej opisanej struktury zależności pomiędzy komponentami, prowadząc do opracowania modelu precyzyjniej pasującego do wymagań użytkowników.

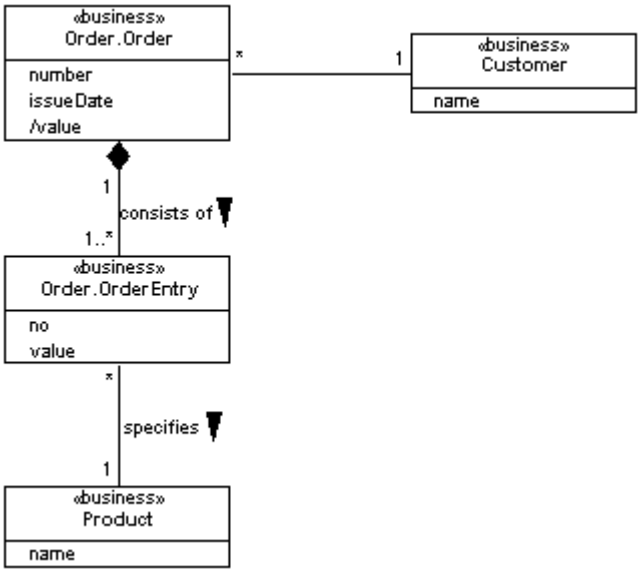
Przykład diagramu komponentów biznesowych został zaprezentowany na rysunku 6. Komponent Customer jest zależny od komponentu Order. Każdy z komponentów ma określone usługi, opisane interfejsami, odpowiednio **ICustomer, IPerson, CustomerMng** oraz **OrderMng, IOrder** i **IOrderEntry**.



Rysunek 6. Przykładowy diagram komponentów biznesowych

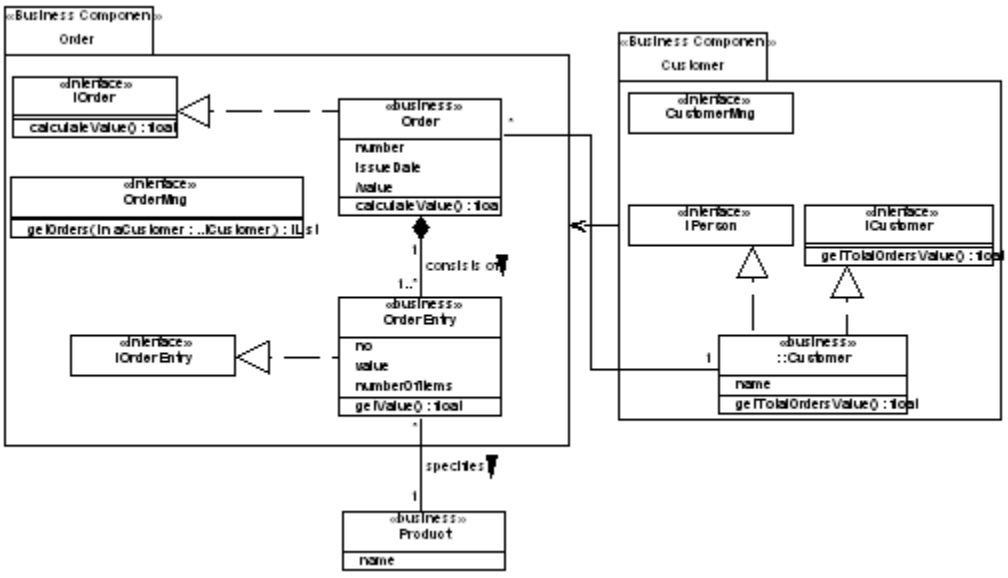
Kolejnym etapem procesu jest określenie zawartości informacyjnej systemu, która następnie powinna znaleźć odzwierciedlenie w zawartości komponentów składających się na rozwiązanie. Podstawowym źródłem informacji, jaka powinna być reprezentowana w systemie są model procesów biznesowych (specyfikujący informacje wykorzystywane przez organizację do sprawnego funkcjonowania) oraz model przypadków użycia, opisujący, jakie informacje, spośród wszystkich wymienionych w modelu procesów, powinny znaleźć się w systemie. Powiązanie przypadków użycia z procesami biznesowymi w znacznym stopniu ułatwia dostęp do potrzebnej informacji. Rysunek 7 przedstawia przykład modelu informacyjnego. Prezentacja informacji jest dokonana z użyciem pojęć dostępnych w ramach modelowania klas i obiektów w notacji *UML*. Oprócz statycznych cech, reprezentowanych przez atrybuty klas możliwe jest pokazanie zależności (reprezentowanych przez związki łączące klasy) występujących pomiędzy bytami obecnymi w tym wycinku rzeczywistości, który jest istotny z punktu widzenia tworzonego systemu.

W kolejnym kroku omawianego etapu należy połączyć ze sobą model komponentów z modelem informacyjnym. Czynność ta pozwala na określenie zawartości informacyjnej komponentu, a tym samym weryfikację możliwości realizacji wymagań, które z nim zostały skojarzone. Łącząc model informacyjny z modelem komponentów należy zwracać szczególną uwagę na gęstość powiązań łączących klasy wchodzące w skład komponentu z klasami znajdującymi się w innych komponentach. Komponenty powinny być projektowane w taki sposób, by liczba powiązań klas w nich zawartych z klasami znajdującymi się na zewnątrz komponentu była jak najmniejsza.



Rysunek 7. Przykładowy model informacyjny

Rysunek 8. przedstawia model komponentów uzupełniony o klasy zidentyfikowane w modelu informacyjnym. Wartym podkreślenia jest fakt istnienia niewielu związków międzykomponentowych łączących klasy.

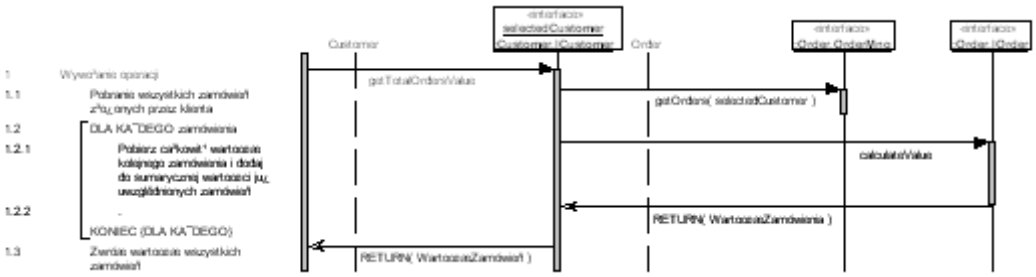


Rysunek 8. Model komponentów uzupełniony o dane modelu informacyjnego

Model informacyjny jest najczęściej podstawą do późniejszej konstrukcji modelu relacyjnej bazy danych, która posłuży do zapewnienia trwałości obiektom powstającym w trakcie pracy systemu.

Statyczne zależności opisane na diagramach komponentów są „konsumowane" w trakcie modelowania interakcji komponentów, obrazujących wzajemne odwoływanie się do usług składowych systemu. Rysunek 9. przedstawia przykładową sytuację, w której realizacja usługi **getTotalOrdersValue** interfejsu **Customer.ICustomer** odwołuje

się do usług komponentów **Order.OrderMng** i **Order.IOrder**. Granice odpowiedzialności komponentów zostały zaprezentowane przy pomocy linii podziału architektury (ang. *architectural boundary*). Usługa **getTotalOrdersValue** odwołuje się do usługi **getOrders** komponentu Order celem pobrania wszystkich zamówień złożonych przez wskazanego klienta (**selectedCustomer**). W efekcie tego wywołania zwracana jest lista (**IList**) elementów typu **Order.IOrder**. Następnie na każdym zamówieniu wywoływana jest usługa **calculateValue**, celem wyliczenia wartości wszystkich złożonych zamówień.

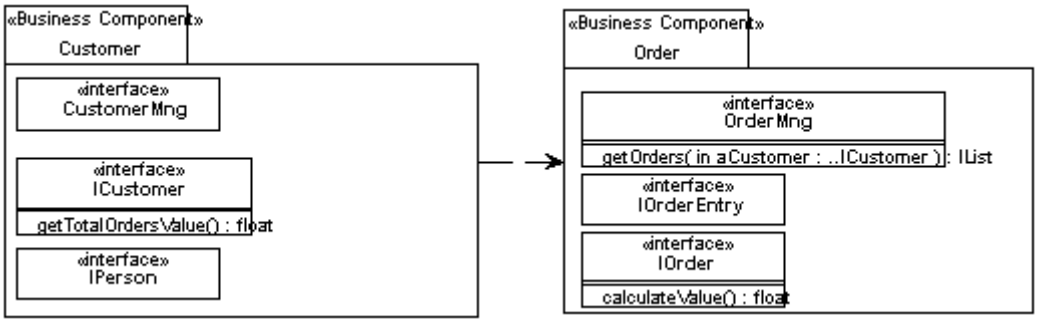


Rysunek 9. Przykładowy diagram sekwencji

Precyzowanie usług komponentów poprzez specyfikacje wzajemnych interakcji jest bardzo wydajną techniką prowadzącą do określania minimalnego zbioru usług, jakie komponenty powinny oferować, by mogły stać się częścią składową konstruowanego rozwiązania. Rysunek 10. prezentuje znany już diagram komponentów biznesowych wzbogacony o deklaracje operacji każdego z interfejsów. Istotne jest to, że istnienie widocznych operacji jest uzasadnione koniecznością wywołania ich, na co najmniej jednym diagramie sekwencji utworzonym w projekcie.

Identyfikowanie operacji (usług) komponentów powinno brać pod uwagę wymagania i ograniczenia sprecyzowane w już zrealizowanych etapach cyklu życia systemu. Elementami, które będą wpływały na usługi są:

- *reguły biznesowe*. Wpływają one na warunki wejściowe oraz wyjściowe usług. Reguły biznesowe są, bowiem inwariantami w działalności organizacji i żadna zmiana stanu systemu nie może wprowadzać systemu w stan, który będzie sprzeczny ze zdefiniowanym zbiorem reguł.
- *architektura techniczna*. Opracowana architektura techniczna może wpływać na zestaw parametrów, które powinny być przesłane razem z wywołaniem określonej usługi.
- *model informacyjny*. Wpływa na listę parametrów definiowanych usług.



Rysunek 10. Diagram komponentów wzbogacony o deklaracje operacji

Określenie precyzyjnych wymagań wobec komponentów umożliwia podjęcie kroków, mających na celu ewentualne pozyskanie komponentów spełniających opisane wymagania. Pierwszym źródłem ich pozyskiwania powinno być korporacyjne repozytorium, przechowujące efekty wcześniejszych prac oraz komponenty pozyskane od dostawców w trakcie realizacji innych projektów. Jeżeli uda się znaleźć komponenty zgodne z określonymi wymaganiami, wówczas ich specyfikacja jest importowana do modelu, zastępując lub wzbogacając istniejącą specyfikację. W przypadku braku komponentów nadających się do realizacji postawionych wymagań, należy podjąć kroki zmierzające do jego pozyskania. Zakup gotowego komponentu, rozszerzenie funkcjonalności istniejących komponentów o pożądaną funkcjonalność, budowa komponentów własnymi środkami, zlecenie budowy komponentu firmie zewnętrznej mogą być środkami do realizacji zadania. Bez względu na przyjętą strategię, proces dostarczania rozwiązania przewiduje negocjacje pomiędzy zespołem konsumującym komponenty i zespołem odpowiedzialnym za ich dostarczenie.

Efektem realizowanych w ramach tego etapu prac jest powstanie produktów pośrednich, które będą wykorzystywane w trakcie dalszych prac nad systemem:

- *model architektury dziedzinowej,*
- *model informacyjny komponentów dziedzinowych,*
- *specyfikacja testów komponentów (oferowanych przez nie usług).*

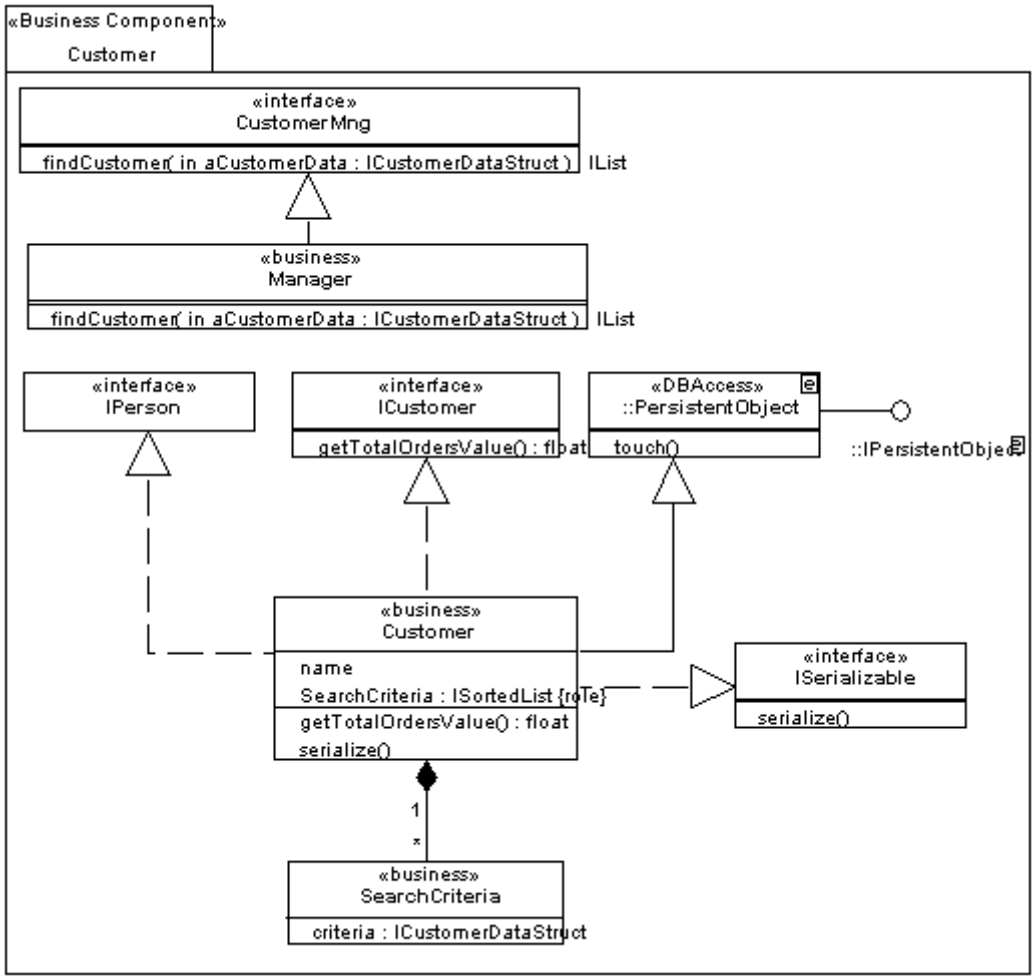
Ostatnim z trzech omawianych etapów procesu dostarczania rozwiązania jest **dostarczenie rozwiązania**. W jego efekcie, wyspecyfikowane wymagania są przekształcane w gotowy system.

Na dostarczenie rozwiązania składają się czynności prowadzące do:

- *opracowania projektu rozwiązania* (uwzględniającego uzgodnioną architekturę techniczną) w oparciu o przyjęty, przyrostowy plan dostarczania,
- *utworzenia i zatwierdzenia prototypu interfejsu użytkownika*,
- złożenia rozwiązania
- akceptacji rozwiązania.

Opracowanie projektu rozwiązania prowadzi do rozszerzenia modelu dziedzinowego o szczegóły wymagane do rozpoczęcia przekształcania modelu w kod aplikacji. Innymi słowy, projekt jest uzupełniany o elementy nie pochodzące z dziedziny problemu, lecz

wynikające z przyjętych ustaleń dotyczących architektury technicznej oraz specyfiki ustalonego środowiska wytwórczo-uruchomieniowego. Stosowanie sprawdzonych wzorców projektowych także wpływa na sposób modyfikacji modelu dziedzinowego.



Rysunek 11. Przykład architektury rozwiązania

Przykład modelu uwzględniającego opisane rozszerzenia został zaprezentowany na rysunku 11. Klasa **Customer** została wyprowadzona z bazowej klasy **Persi-stentObject**, implementującej interfejs **IPersistentObject**. Z racji tego, że klasy te pochodzą z bibliotek zewnętrznych w stosunku do tworzonego modelu, zostały zaznaczone jako **ExternalClass** – czyli nie mogą być modyfikowane (a jedynie wykorzystywane w postaci takiej, jakie są) w modelu tworzonego rozwiązania. Ponadto, klasa **Customer** została rozbudowana o zależność z interfejsem **ISerializable**, w efekcie została w niej zadeklarowana operacja **serialize**(). Model prezentuje także sposób implementacji związku przy pomocy atrybutu **SearchCriteria {role}** typu **ISortedList**.

Kolejnym elementem etapu jest napisanie kodu systemu pozwalającego na połączenie ze sobą komponentów wchodzących w skład rozwiązania, elementów interfejsu użytkownika oraz ewentualnej bazy danych. W dalszej kolejności występują testy poszczególnych elementów systemu, testy integracyjne i akceptacja systemu (bądź przyrostu).

Głównymi produktami etapu są:

- *prototyp interfejsu użytkownika,*
- *model architektury rozwiązania,*
- *model rozwiązania,*
- *funkcjonująca aplikacja,*
- *środowisko instalacyjne.*

Pozostałe trzy etapy dzielą się na dwie grupy zadań. Planowanie przyrostów dotyczy tych systemów, których pełne wdrożenie dzielone jest na mniejsze etapy. Druga grupa dotyczy wdrożenia (instalacji oprogramowania w środowisku docelowym, transformacja danych, itp.) z systemów zastanych oraz późniejszej jego pielęgnacji (obsługa błędów, zmian wymagań, itp.).

Ostatnim procesem składającym się na podejście *Supply – Manage – Consume* jest proces zarządzania komponentami (por. rysunek 13). Uporządkowane podejście do wytwarzania systemów wymusza także uporządkowanie procesu zarządzania posiadanymi zasobami, mogącymi znaleźć zastosowanie w wielu projektach. Obecna praktyka współdzielenia zasobów najczęściej sprowadza się do mniej lub bardziej sformalizowanej wymiany informacji w ramach poszczególnych zespołów. Podejście takie nie sprawdza się jednakże w przypadku prób rozwiązania tego problemu w skali całej firmy. Zmiana kultury współdzielenia wiedzy/zasobów wymaga uwzględnienia poniższych aspektów:

- *procesu zarządzania zasobami (komponentami),*
- *planu wdrożenia procesu w firmie,*
- *narzędzi, których zadaniem będzie wsparcie procesu.*

Dzięki temu możliwe będzie uniknięcie typowych pułapek, w jakie można wpaść wdrażając proces zarządzania komponentami w firmie:

- brak możliwości efektywnego przeszukania katalogu zasobów. Źle, bądź niejednolicie, opisane elementy katalogu będą stanowiły zasób firmy, do którego dostęp będzie praktycznie niemożliwy. Wdrażany proces powinien w jasny sposób precyzować zasady publikowania komponentów oraz umożliwić efektywne ich wyszukiwanie.
- publikowanie komponentów o złej jakości lub nieopisanej funkcjonalności i sposobie stosowania. Polityka jakości powinna być na stałe wpleciona w cały proces zarządzania elementami wielokrotnego użytku. Dostarczanie produktów o złej jakości będzie prowadziło do zmniejszenia motywacji do podjęcia wysiłku związanego z utrzymywaniem oraz wykorzystywaniem wspólnej bazy zasobów.
- ignorowanie problemu zarządzania konfiguracją. Korporacyjne bazy komponentów z upływem czasu będą zasilane kolejnymi wersjami stosowanych komponentów. Zarządzanie konfiguracją umożliwi sprawną wymianę wersji oraz pozwoli na uniknięcie pułapek związanych z niezgodnością interfejsów w kolejnych wersjach komponentów.

Proces zarządzania komponentami rozpoczyna się od etapu pozyskania komponentu.

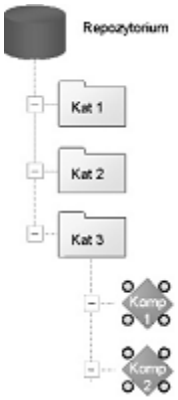
Pozyskanie komponentu powinno brać pod uwagę m.in.:

- zgodność z wymaganiami funkcjonalnymi oraz
- zgodność z przyjętą architekturą techniczną.

Sposobem pozyskania komponentu może być zarówno zakup na zewnętrznym rynku jak i zakontraktowanie jego wyprodukowania na zlecenie. Alternatywą dla zakupu komponentu jest jego wypożyczenie (gotowe pakiety typu SAP) lub wydzierżawienie na określony okres (np. *Web Services*). Katalizatorem tak dużej swobody wyboru jest wyraźny rozdział pomiędzy specyfikowaniem wymagań wobec komponentu i późniejsza dostawą gotowego elementu zgodnego z tymi wymaganiami.

Pozyskany komponent powinien następnie być poddany procesowi certyfikacji, w trakcie, którego zostanie sprawdzona jego jakość. W zależności od przyjętego schematu certyfikacji, działalność ta może być mniej lub bardziej sformalizowana.

Komponent zgodny z przyjętymi normami jakości jest następnie publikowany w repozytorium zgodnie z ustalonymi zasadami (przykład katalogu przedstawia rysunek 12).



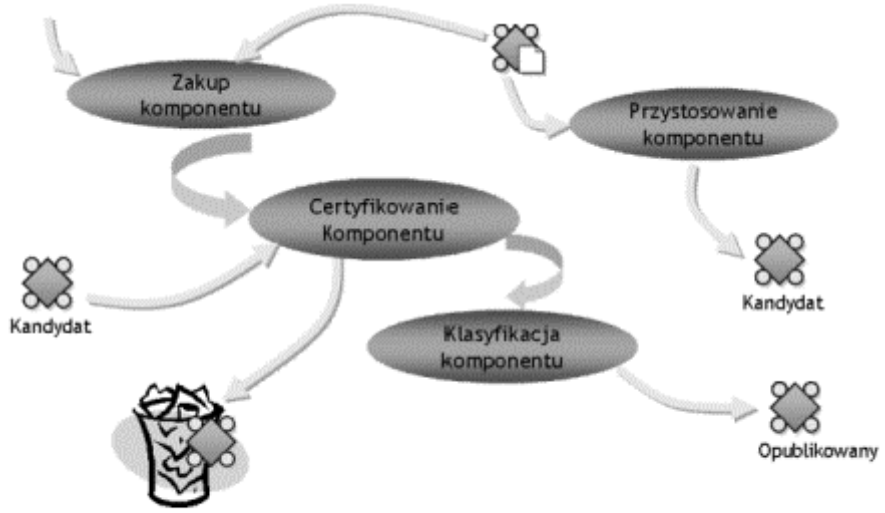
Rysunek 12. Katalog komponentów

Wartym rozważenia jest uwzględnienie poniżej podanych cech, których wartości pozwolą na późniejsze efektywne jego wyszukiwanie:

- *stosowana technologia*. Wartość tej cechy powinna wskazywać na standardowe technologie i platformy, na których komponent może być stosowany (np. .NET, JavaBeans, COM),
- *architektura.* Wartość cechy powinna wskazywać na warstwę aplikacji, w której komponent powinien być stosowany (np. komponent dziedzinowy, komponent interfejsu użytkownika, itp.)
- *funkcjonalność.* Wartość cechy powinna umożliwić klasyfikację.

Istotnym elementem publikacji komponentów jest późniejsze zarządzanie kolejnymi ich wersjami. Analiza wpływu zmiany wersji komponentu (zarządzanie wersjami zarówno specyfikacji jak i komponentów winno być zapewnione przez repozytorium, w którym są składowane komponenty) na nowszą w systemach, w których został on zastosowany, wymaga porównania specyfikacji usług komponentów. Automatyczne powiadamianie zainteresowanych stron o pojawieniu się nowych wersji komponentów, wymaga z kolei utrzymywania informacji o kolejnych zastosowaniach komponentu.

Posiadanie spójnego i dobrze opisanego katalogu komponentów wymagane jest po to, by użytkownik mógł efektywnie wyszukiwać potencjalne składowe tworzonych systemów. Należy pamiętać o tym, że znajdujący się w repozytorium opublikowany komponent z punktu widzenia kolejnego zastosowania jest jedynie kandydatem. Późniejsza ewaluacja oraz rozważanie możliwych alternatyw jest typową czynnością wykonywaną w ramach procesu Supply – Manage – Consume, czyniąc proces zarządzania komponentami kluczowym elementem całego łańcucha działań.

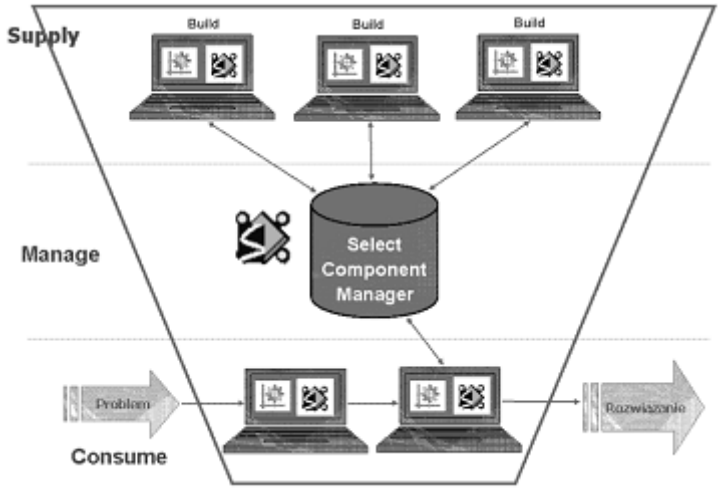


Rysunek 13. Proces zarządzania komponentami

5. Select Perspective w praktyce

Zastosowanie procesu produkcyjnego wymaga utworzenia odpowiedniego środowiska umożliwiającego zaangażowanie wszystkich osób biorących udział w realizacji przed-sięwzięć. Autorzy metodyki (firma Select Busines Solutions Ltd. – www.selectbs.com [http://www.selectbs.com/]) zadbali o odpowiednie wsparcie dla metodyki dostarczając linię produktów Select Component Factory™. W jej skład wchodzą dwa podstawowe produkty: Select Component Architect™ oraz Select Component Manager™. Select Component Architect jest narzędziem służącym do modelowania aplikacji z wykorzystaniem notacji UML (z rozszerzeniami umożliwiającymi modelowanie procesów biznesowych oraz struktury relacyjnej bazy danych). Z punktu widzenia omawianego powyżej procesu wytwórczego, Select Component Architect znajduje zastosowanie zarówno w procesie dostawy komponentów (projektowanie i budowa składowych aplikacji) jak i dostarczania gotowego rozwiązania (projekt i budowa finalnego produktu dla klienta końcowego, wykorzystującego komponenty). Select Component Manager jest wykorzystywany przede wszystkim przez proces zarządzania komponentami (katalogowanie komponentów) oraz jako medium wymiany informacji pomiędzy pozostałymi dwoma procesami (specyfikowanie wymagań, dostarczanie kandydatów na komponenty korporacyjne, pobieranie komponentów). Zależności pomiędzy składowy-

mi linii Select Component Factory a procesem produkcyjnym przedstawia rysunek 14.



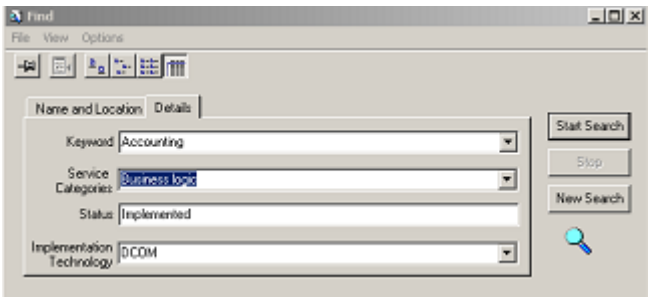
Rysunek 14. Produkty linii Select Component Factory a model Supply-Manage-Consume

W rozdziale 3. *Korzyści, jakich oczekuje się od technologii komponentowej* sygnalizowano konieczność powstania ogólnodostępnego rynku gotowych prefabrykatów, z których późniejsze aplikacje będą mogły być składane. Jednym z wymogów umożliwiających powstanie takiego rynku jest stworzenie odpowiedniej infrastruktury, udostępniającej zarówno katalogi komponentów jak i dającej możliwość dokonania ich zakupu. Twórcy metodyki Select Perspective dostarczają takiej infrastruktury w postaci narzędzia Select Component Manager (patrz rysunek 14).

Informacje dostępne w narzędziu Select Component Manager (dalej zwany także SCM) można podzielić na dwie podstawowe kategorie:

- elementy znajdujące się w repozytorium korporacyjnym,
- elementy dostępne na „wolnym rynku".

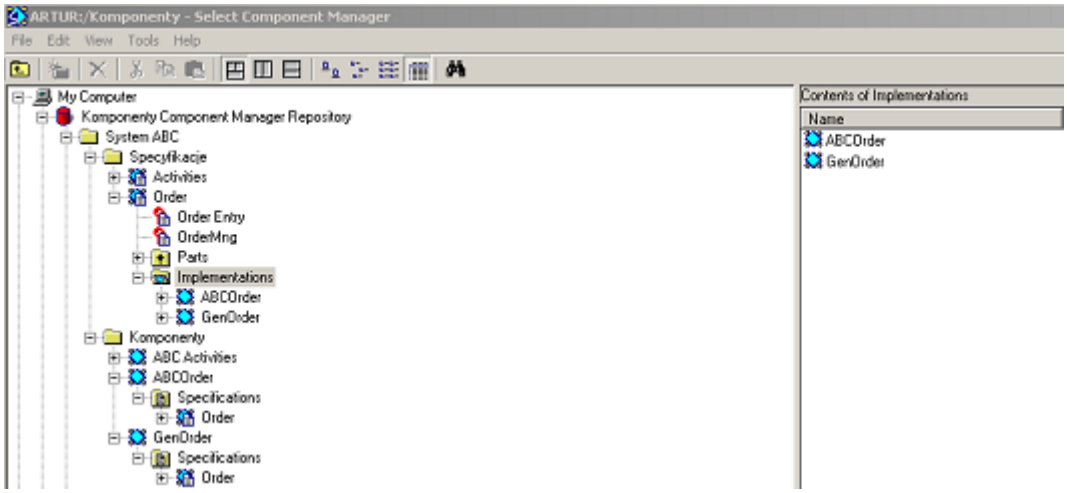
Elementy znajdujące się w repozytorium są udostępniane wybranym osobom i mogą być pobrane do wykorzystania z użyciem narzędzia Select Component Manager (lub Select Component Portal™ - aplikacji przeglądarkowej pozwalającej na dostęp do repozytorium przez Internet).



Rysunek 15. Wyszukiwanie komponentów

Jedną z podstawowych czynności realizowanych na repozytorium jest wyszukiwanie elementów zgodnie z określonymi kryteriami (patrz rysunek 15). Oprócz nazwy oraz

miejsca przeszukiwania, możliwe jest podanie dodatkowych informacji, zawężających zakres poszukiwanej informacji, np. technologii, w której komponent został wykonany, czy też warstwy aplikacji, w skład której powinien wchodzić.



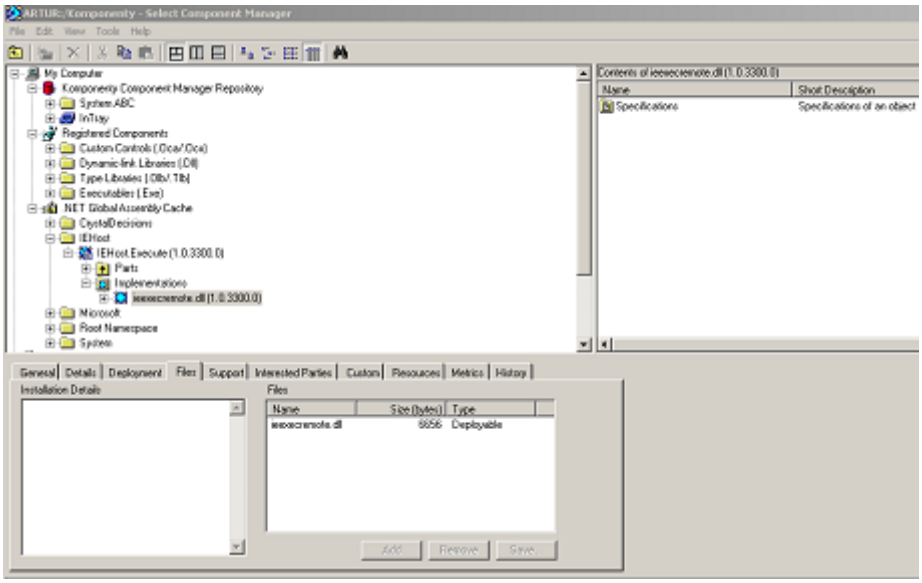
Rysunek 16. Komponenty i specyfikacje w Select Component Manager

Komponenty znajdujące się w repozytorium są skatalogowane według określonych w organizacji zasad oraz posiadają wyraźnie wyróżnioną część specyfikacji oraz związane z nią elementy realizujące. Rysunek 16. przedstawia przykład opublikowanej specyfikacji komponentu Order (katalog SystemABC/Specyfikacje) oraz dwóch jego implementacji GenOrder oraz ABCOrder (katalog SystemABC/Komponenty). Na specyfikację komponentu Order składają się dwa interfejsy: OrderEntry oraz OrderMng.

Select Component Manager daje także dostęp do komponentów zarejestrowanych (czyli zakupionych i użytkowanych) na komputerze, na którym uruchomiony jest SCM. Informacje o komponentach są pobierane z rejestru systemu MS Windows oraz rejestru platformy Microsoft .NET (*.NET Global Assembly Cache*). Przykład prezentacji tych informacji przedstawia rysunek 17. Informacje odczytywane z rejestru MS Windows są prezentowane w podziale na poszczególne typu komponentów:

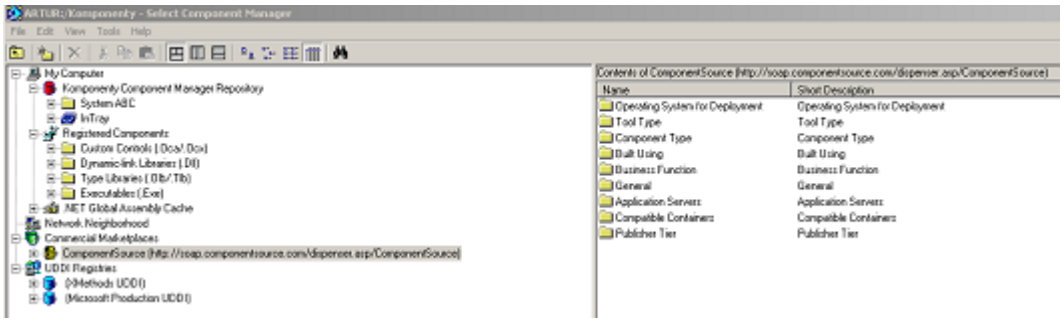
- kontrolki interfejsu graficznego użytkownika (oca, ocx),
- biblioteki dynamiczne (.dll),
- biblioteki typów (.tlb) oraz
- pliki wykonywalne (*exe).

Podobnie podzielone są komponenty odczytane z rejestru platformy Microsoft .NET.



Rysunek 17. Komponenty zarejestrowane na stacji roboczej

Drugą kategorią informacji dostępnych w narzędziu Select Component Manager (patrz rysunek 18) są prezentacje specyfikacji komponentów dostępnych na rynku. Źródłem informacji jest największy obecnie portal oferujący komponenty Component Source (www.componentsource.com) oraz zarejestrowane w rejestrach UDDI serwery oferujące usługi internetowe (ang. *Web Service*). Dzięki tej funkcjonalności możliwe jest przeszukiwanie opublikowanych na rynku ofert celem ewentualnego wykorzystania oferowanych produktów w realizowanym przedsięwzięciu. Wartym podkreślenia jest także fakt pełnej integracji Select Component Manager z Select Component Architect (narzędzie do modelowania z użyciem notacji UML). Integracja ta umożliwia automatyczny import specyfikacji komponentu do modelu UML, dzięki czemu bardzo szybko można zamodelować sposób wykorzystania komponentu w budowanym systemie. Funkcjonalność ta jest szczególnie użyteczna na etapie pielęgnacji systemu – zarządzanie kolejnymi wersjami komponentu wymaga stałej aktualizacji informacji o oferowanych przezeń usługach oraz ewentualnego powiadamiania o niezachowaniu zgodności „w dół".



Rysunek 18. Komponenty dostępne na "wolnym rynku"

Metodyka Select Perspective jest opisana i dostarczana w postaci elektronicznej w narzędziu **Process Director**™. Taka postać prezentowania informacji daje możliwość

łatwej jej aktualizacji oraz dystrybucji pomiędzy członkami zespołów. Opis metodyki nie ogranicza się jedynie do przedstawienia poszczególnych etapów procesu produkcyjnego. W jego skład wchodzi m.in. zarządzanie ryzykiem przedsięwzięcia, definicja produktów poszczególnych faz, definicja ról członków zespołów, odwołania do narzędzi informatycznych, mogących wesprzeć dany etap procesu produkcyjnego, odniesienia do proponowanych technik pracy.

6. Podsumowanie

Metodyka Select Perspective jako jedyna z dostępnych na rynku jest dostarczana wraz z pełnym wsparciem narzędziowym pozwalającym na wdrożenie komponentowego procesu wytwórczego w organizacjach wytwarzających systemy informatyczne. Leżąca u podstaw metodyki filozofia podziału prac przy produkcji systemów na dostawców i odbiorców oraz jasne sprecyzowanie zakresu odpowiedzialności procesu zarządzania komponentami pozwala na dostosowywanie jej do specyfiki działań wdrażających ją organizacji. Dzięki mechanizmom rozszerzającym (ang. *extension mechanism*) języka Unified Modeling Language możliwe staje się jej przystosowanie do środowiska wytwórczego nie w pełni implementującego założenia technologii obiektowej oraz komponentowej z zachowaniem promowanej przez firmę Select Business Solutions filozofii pracy.

Bibliografia

[ALLEN1998] P Allen i S Frost, *Component-Based Development for Enterprise Systems, Applying the Select Perspective*, Cambridge University Press, 1998.

[APP2003] H Apperly, *Service- and Component-based Development*, Addison-Wesley, 2003.

[APPICBD] Hedley Apperly, *Introducing Component-Based Development*, http://www.selectbs.com/education/whitepapers.htm.

[HEIN2001] G. T. Heineman i W. T. Councill, *Component-Based Software Engineering. Putting the Pieces Together*, Addison-Wesley, 2001.

[SBSISP] *Introducing Select Perspective*, http://www.selectbs.com/education/whitepapers.htm.

[SBSSRROI] *Software Reuse ROI*, http://www.selectbs.com/education/whitepapers.htm.