XDE.NET jako skuteczne środowisko wytwarzania aplikacji webowych dla systemów mobilnych

Andrzej STASIAK[1], Michał WOLSKI[2]

[1)] Zakład Systemów Komputerowych, Instytut Teleinformatyki i Automatyki WAT,
ul. Kaliskiego 2, 00-908 Warszawa
[2)] Instytut Informatyki Akademii Podlaskiej
ul. Sienkiewicza 51, 08-110 Siedlce

STRESZCZENIE: W artykule przedstawiono opis pakietu narzędziowego VS.NET (Microsoft) z Rational XDE (IBM) do wytwarzania aplikacji webowych, pracujących w środowisku urządzeń mobilnych na platformie .NET (firmy Microsoft) wraz z metodyką jego użycia, bazującą na procesie wytwórczym Rational Unified Process. Opisano także metodykę tworzenia projektów w XDE. Szczególną uwagę poświęcono nowym funkcjom środowiska XDE, pozwalającym na badanie własności behawioralnych aplikacji oraz dającym możliwość wykonywania testów systemu już na poziomie specyfikacji języka UML. Opis procesu wytwórczego zilustrowano na przykładzie aplikacji z ograniczeniami czasowymi, zaimplementowanej w ASP.NET, a przeznaczonej na platformę PocketPC.

1. Wprowadzenie

Systemy reaktywne stanowią ten rodzaj systemów, których działanie jest uwarunkowane czasowo i znane są także jako systemy czasu rzeczywistego (ang. *Real-Time Systems*) [2], [15]. Systemy RT zazwyczaj działają na pokładach „nosicieli", jako systemy wbudowane (ang. *Embedded Systems*), które są tworzone pod konkretne rozwiązania techniczne (np. systemy nawigacji satelitarnej GPS, systemy sterowania rakietą, urządzeniem peryferyjnym komputera czy nowoczesnym pojazdem kołowym (ASP, ESP) itd.).

Systemy RT dotychczas były i są nadal budowane przy pomocy dedykowanych dla tego typu rozwiązań środowisk komputerowego wspomagania projektowania CASE (np. I-Logic Rhapsody, Rational Suit Real-Time Development Studio, TimeWiz). Wspomniane narzędzia, choć skutecznie wspierają budowę systemów reaktywnych, nie wspomagają procesów wytwórczych systemów dedykowanych na platformę webową. Z tego też względu celowe jest znalezienie takiego środowiska programistycznego, które pozwoli na wykonanie projektu i implementacji webowej aplikacji RT.

Obecnie jednym z wiodących środowisk wspomagających implementację aplikacji webowych jest Visual Studio .NET (środowisko RAD firmy Microsoft, stanowiące Lower CASE). Pominięcie w nim możliwości tworzenia metamodelu (niezwiązanego z konkretną implementacją) stanowi jego zasadniczą wadę. Bezpośrednio Visual Studio .NET nie wspiera projektowania aplikacji, np. w UML, jak ma to miejsce w Rational Suit Real-Time Development Studio [15]. Dlatego potrzebne jest narzędzie, które uzupełni ten istotny brak. Takim środowiskiem (typu Upper CASE) jest IBM Rational XDE Developer Plus (XDE = e**X**tended **D**evelopment **E**nvironment), które rozszerza Visual Studio o możliwość projektowania aplikacji zarówno poprzez modele, jak i odpowiadający im kod [5], [7], [9] i [10]. Wskazany zestaw narzędziowy wymaga precyzyjnego określenia sposobu jego stosowania, aby wynikiem jego użycia było oprogramowanie dla webowych systemów mobilnych.

2. Proces wytwórczy

Systemy czasu rzeczywistego wymagają od projektantów dedykowanego procesu wytwórczego, który szczególną uwagę zwracałby na aspekty czasowe tworzonych rozwiązań. W procesie tym zakłada się, że wytwarzanie aplikacji, już od momentu planowania aż do jej implementacji, powinno być wykonane z uwzględnieniem dobrych praktyk inżynierskich, które wymuszają zastosowanie sprawdzonej wcześniej metodyki wytwarzania oprogramowania.

Jedną z dostępnych na rynku interesujących propozycji jest Rational Unified Process (RUP), który jest zunifikowanym procesem wytwórczym oprogramowania, dostarczającym praktycznych wskazówek, wzorców dokumentów i narzędzi, szablonów dokumentów oraz przykładów postępowania dla niemalże wszystkich działań związanych z procesem wytwarzania oprogramowania [15].

RUP jest określony jako programowa metoda ewolucyjna, która wspiera iteracyjne wytwarzanie oprogramowania, sterowane przez architekturę, ukierunkowane na przypadki użycia [8].

Iteracyjność w RUP (rys. 1) oznacza wytwarzanie systemu przez jego ulepszanie w wielu cyklach procesu. Każda bieżąca iteracja kończy się opracowaniem uruchamialnego fragmentu kodu programu bądź systemu, skutecznie prowadząc do jego wdrożenia, przy czym warunkiem przejścia do kolejnej iteracji jest uzyskanie pozytywnego wyniku analizy ryzyka [8], prowadzonej w ramach zarządzania projektem (rys. 2).

Rys. 1. Iteracyjny proces rozwoju oprogramowania [13]

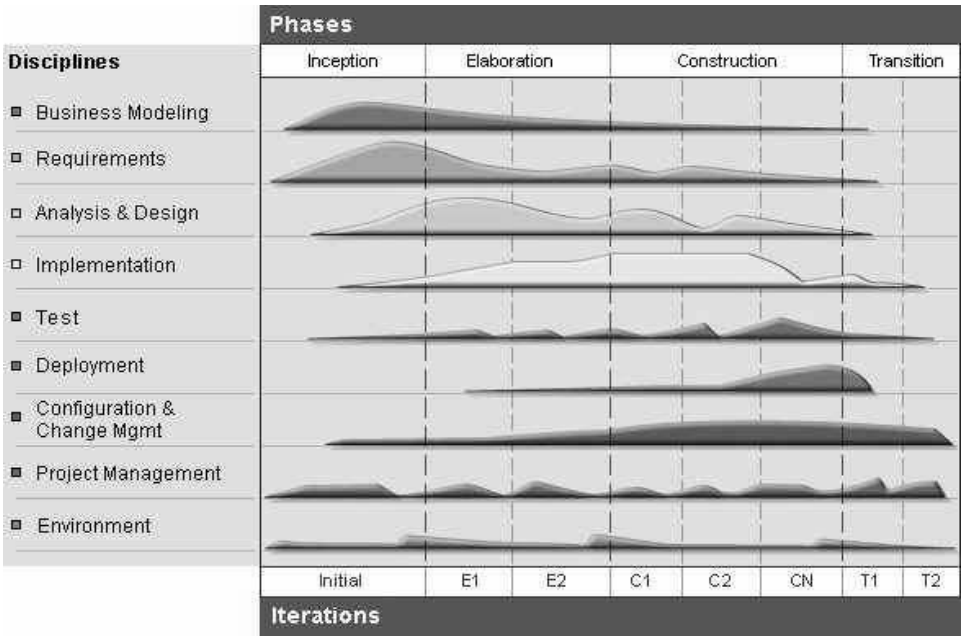
Zorientowanie na architekturę w RUP oznacza, że wybór architektury oprogramowania ma zasadnicze znaczenie dla procesu tworzenia oprogramowania. Dodatkowo, proces ten sterowany jest przez przypadki użycia, które określają usługi systemu, jego funkcjonalność, która powinna być zgodna z wymaganiami na system.

Dalej w artykule przedstawiono opis procesu wytwórczego RUP (uszczegółowionego, dla analizowanych platform, w dodatku: „Plug-in RUP for Microsoft® .NET"), który, przy założeniu wykorzystania zestawu narzędzi XDE.NET i VS.NET, będzie skutecznie umożliwiał wytwarzanie aplikacji mobilnych na platformie WEB.

2.1. Cykl tworzenia oprogramowania

Cykl wytwarzania oprogramowania jest jednym z najważniejszych elementów wyróżniających metodykę. W Rational Unified Process cykl wytwarzania oprogramowania opisywany jest względem dwóch osi: iteracji

i dyscyplin (rys. 2). W pionie przedstawione są statyczne aspekty wytwarzania oprogramowania, takie jak czynności, role, przepływy oraz artefakty. W poziomie natomiast przedstawione zostały dynamiczne aspekty wytwarzania oprogramowania, takie jak fazy (etapy) oraz iteracje.



Rys. 2. Model cyklu tworzenia oprogramowania [13]

2.2. Fazy tworzenia oprogramowania

RUP składa się z czterech faz wytwórczych oprogramowania. Pierwsza faza to rozpoczęcie (ang. *inception*). W fazie tej określane są założenia: techniczne, rynkowe i ekonomiczne dla projektu. Tworzony jest harmonogram prac oraz szacuje się ryzyko powodzenia projektu. Pod koniec fazy rozpoczęcia następuje określenie celów przedsięwzięcia oraz zapada decyzja, czy nastąpi przystąpienie do pełnego procesu wytwórczego. Zwykle faza ta przebiega w jednej iteracji.

Gdy zapada decyzja o kontynuacji, projekt przechodzi do fazy opracowania (ang. *elaboration*), która przebiega w kilku iteracjach (zwykle więcej niż dwóch – w zależności od złożoności systemu). W fazie tej analizujemy dziedzinę problemu projektowego, definiujemy architekturę systemu, przygotowujemy plan prac i neutralizujemy największe zagrożenia [1].

Wszystkie wymienione czynności mogą zaistnieć tylko w sytuacji gdy opisano większość wymagań na system.

Budowa (ang. *construction*) to kolejna faza, podczas której w kilku iteracjach i przyrostach systemu następuje budowanie oraz integrowanie wszystkich komponentów składających się na tworzony system. Końcowym produktem tej fazy jest gotowy, w pełni przetestowany oraz udokumentowany system nadający się do wdrożenia.

Ostatnią fazą procesu wytwórczego RUP jest przekazanie (ang. *transition*). Jej celem jest przekazanie użytkownikowi końcowemu gotowego systemu (zwykle w jednej iteracji). Podczas fazy przekazania na światło dzienne wychodzą nieprzewidziane do tej pory problemy z systemem, które często wymagają dodatkowych prac programistycznych. Po tych czynnościach, mających na celu ostateczne dopracowanie rozwiązania, może nastąpić wdrożenie systemu, które kończy proces wytwarzania oprogramowania.

Wymienione powyżej fazy składają się z iteracji, czyli działań produkcyjnych, zmierzających do wytworzenia produktów. Przejście przez wszystkie etapy stanowi cykl wytwarzania oprogramowania. W wyniku przejścia przez wymienione fazy, tworzona jest nowa generacja oprogramowania.

3. Zintegrowane środowisko wytwórcze

Zintegrowane środowisko wytwórcze Visual Studio .NET z IBM Rational XDE łączy w sobie cechy narzędzi Upper i Lower CASE, które umożliwiają zarówno analizę, projektowanie, implementację, jak i uruchamianie aplikacji (na przykład webowych) bez potrzeby przełączania się między nimi (tj. bez konieczności przechodzenia pomiędzy heterogenicznymi interfejsami i repozytoriami środowisk wytwórczych). Poniżej, w tabeli 1, został zaprezentowany wykaz czynności projektowych z uwzględnieniem podziału na poszczególne narzędzia.

Tab. 1. Wykorzystanie narzędzi pakietu XDE podczas cyklu wytwórczego oprogramowania

Dyscyplina (zakres procesu)	Czynność	Narzędzie	Zakres zastosowania
Wymagania	Tworzenie wymagań na system	RequisitePro	Gromadzenie i utrzymywanie wymagań funkcjonalnych i niefunkcjonalnych w repozytorium (modeli opisowych).

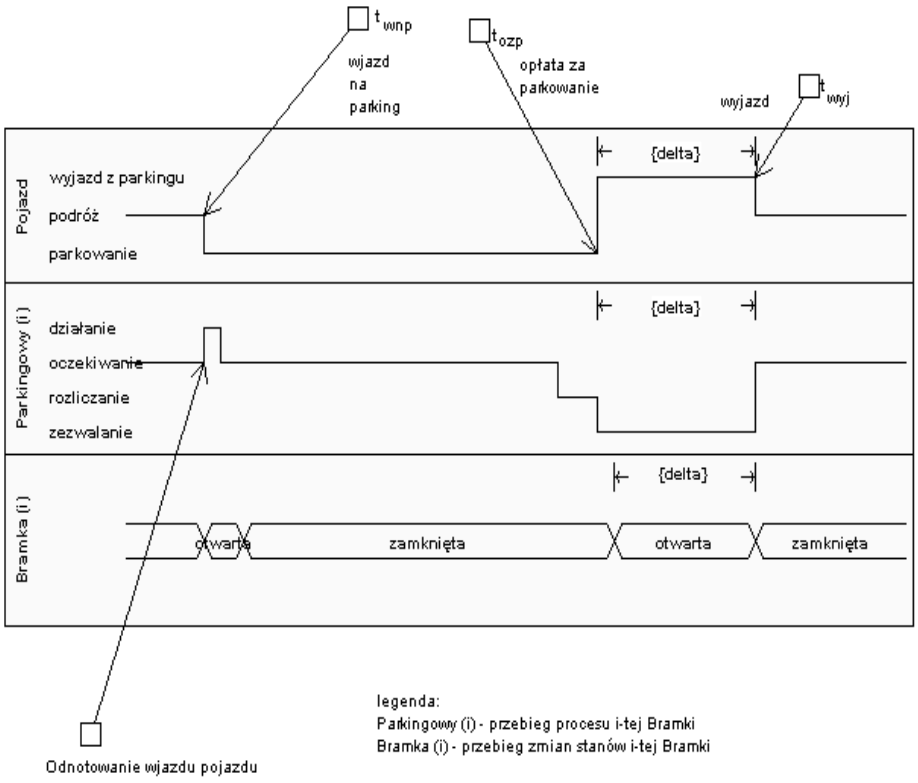
Dyscyplina (zakres procesu)	Czynność	Narzędzie	Zakres zastosowania
	Modelowanie usług	XDE	Modelowanie przypadków użycia w UML - stanowi niezależne repozytorium wymagań funkcjonalnych, które może zostać zsynchronizowane z repozytorium RequisitePro.
Analiza i projektowanie	Modelowanie architektury	XDE	Definiowanie architektury systemu.
	Modelowanie projektu	XDE	Uszczegółowienie modeli oraz opracowanie modeli tworzących widok komponentów systemu.
	Modelowanie bazy danych	XDE	Tworzenie repozytorium projektu modelu bazy danych (istnieje możliwość synchronizacji na każdym etapie projektowania i implementacji).
Implementacja	Inżynieria w przód (ang. *forward engineering*)	XDE i baza danych	Umożliwia automatyczne przekształcenie utworzonych modeli (wymagań, analitycznych lub projektowych) w szkielety aplikacji lub baz danych.
	Implementacja bazy danych	VS.NET i dowolna baza danych	Umożliwia dostęp i kodowanie baz danych z poziomu środowiska VS.NET.
	Implementacja aplikacji	VS.NET	Budowanie implementacji rozwiązania.
	Inżynieria wstecz (ang. *reverse engineering*)	XDE	Tworzy model systemu na podstawie dostarczonego kodu
	Modelowanie kodu	XDE	Realizowane jest jako wynik inżynierii wstecz lub synchronizacji modeli z implementacją
Testowanie	Debugowanie	VS.NET	Poprawianie błędów w kodzie aplikacji
	Budowanie śladowego diagramu sekwencji	XDE (Visual Trace) i VS.NET	Dokumentowanie aspektów dynamicznych aplikacji w trakcie jej funkcjonowania.
	Testowanie wydajności aplikacji	XDE (Quantify) i VS.NET	Badanie czasu działania poszczególnych metod.
	Testowanie pokrycia kodu testami	XDE (PureCoverage) i VS.NET	Poszukiwanie fragmentów kodu niepokrytego testami.
	Znajdowanie błędów wykonania i błędów pamięci	XDE (Purify) i VS.NET	Poprawianie wydajności aplikacji.

Niewątpliwie, korzyścią dla zespołu projektowego jest fakt, że zintegrowane środowisko wspiera cały proces wytwórczy oprogramowania oraz umożliwia utrzymywanie projektu w jednym repozytorium.

4. Problem projektowy

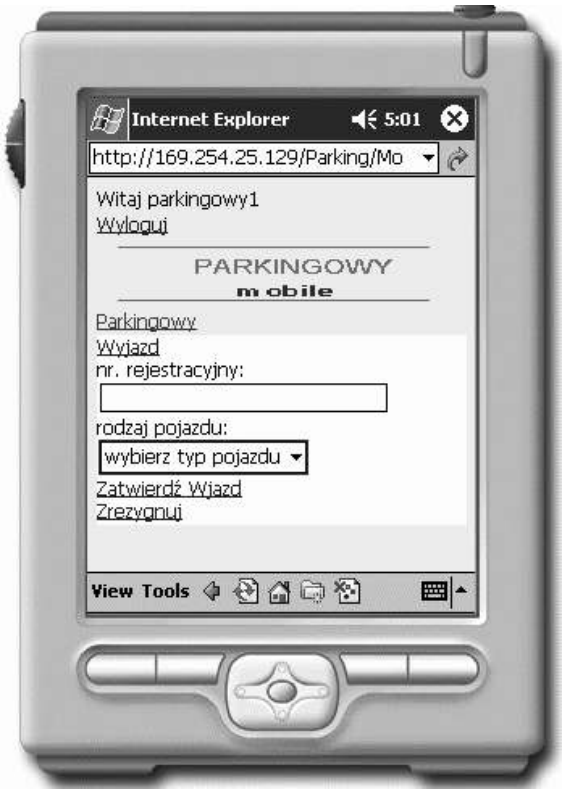
W dalszej części przedstawiono szkic ogólnej metody wytwarzania mobilnych aplikacji webowych, opierając się na przykładzie aplikacji „Parkingowy", tworzonej na platformę urządzeń przenośnych (ang. *mobile devices*) z systemem PocketPC.

System ten jest systemem reaktywnym o cechach „miękkiego" systemu czasu rzeczywistego (ang. *soft RT system*).



Rys. 3. Diagram przebiegów czasowych w systemie „Parkingowy"

Zasadniczym przeznaczeniem przykładowego systemu „Parkingowy" jest wspomaganie obsługi rozliczeń za parkowanie. Z punktu widzenia systemów RT ważne są trzy zdarzenia z procesu parkowania. Pierwszym z nich jest wjazd na parking (t_{wnp}), drugim – moment, od którego następuje opłata za czas parkowania (t_{ozp}), a kolejny czas – delta jest określony jako odcinek czasu, podczas którego pojazd musi opuścić parking. Przekroczenie tego czasu powoduje nieotwarcie bramki wyjazdowej (mimo uiszczenia opłaty).



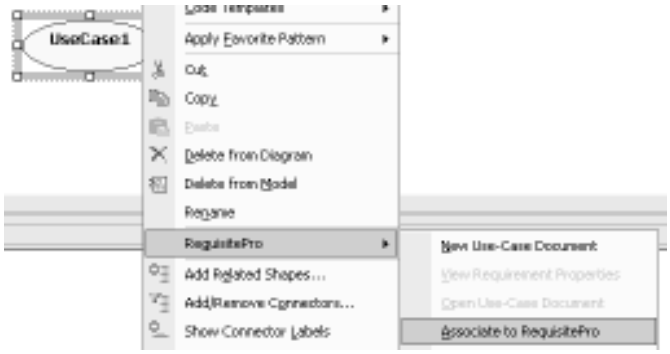
Rys. 4. Zrzut ekranu aplikacji „Parkingowy"

W prezentowanym przykładzie techniczną realizację systemu zaimplementowano na platformie mobilnej PocketPC z wbudowanym systemem łączności radiowej (ang. *wireless*).

Aplikacja została tak zaprojektowana, by opłata za parkowanie była uiszczana przy pojeździe w chwili odjazdu lub przy wejściu kierowcy na teren parkingu. Dodatkowo, aplikacja umożliwia rejestrowanie utargu z dowolnej bramki.

5. Wymagania na system

Zintegrowane środowisko programistyczne wspomaga projektantów systemów już od zbierania wymagań na system. Rational XDE może być zintegrowane z narzędziem dedykowanym do zarządzania wymaganiami – Rational RequisitePro. Wymagania mogą być modyfikowane zarówno w XDE, jak i w RequisitePro, gdyż istnieje pełna synchronizacja pomiędzy tymi narzędziami i ich repozytorium. Dzięki takiej synchronizacji oraz współpracy, przypadki użycia (opisywane za pomocą graficznego modelu Use Case) zostają rozszerzone o informację zawierającą związki z wymaganiami (wyrażonymi za pomocą modeli opisowych). RequisitePro dodatkowo daje nam możliwość śledzenia wszystkich zmian zachodzących w wykreowanych w XDE przypadkach użycia.



Rys. 5. Synchronizacja z RequisitePro - fragment menu podręcznego

Sytuacja, w której wymagania na system są zsynchronizowane z konkretnym przypadkiem użycia, umożliwia zespołom projektowym lepszą kontrolę nad zmianami wymagań oraz weryfikację ich wizji systemu.

Zastosowanie RequisitePro umożliwia łatwe pogrupowanie wymagań przez przypisanie im odpowiedniej wartości priorytetu wykonania (np.: wysoki, średni i niski). Takie usystematyzowanie wymagań (w zależności od ich ważności), zwane metodą Triage [16], pozwala wskazać wymagania do realizacji w pierwszej kolejności i te, które użytkownikowi są potrzebne, ale ich obecność w gotowym systemie nie jest najważniejsza. Dodatkowo, RequisitePro umożliwia tworzenie i prezentację hierarchii wymagań przez określenie związków (rodzic-dziecko) między nimi. Określenie statusu wymagania (np.: nie zrealizowano, zrealizowano, zatwierdzono) umożliwia w każdym momencie życia projektu wskazanie wymagań, które mają już swoją implementację, a także tych, których działanie nie jest jeszcze do końca zaimplementowane lub przetestowane.

Efektywne zarządzanie wymaganiami na system, przez cały cykl wytwarzania oprogramowania, stanowi ogromną zaletę opisywanego środowiska, gdyż pozwala w pełni zautomatyzować zarządzanie zmianami w wymaganiach. Umożliwia również ciągłe obserwowanie postępów prac projektowych.

6. Struktura projektu

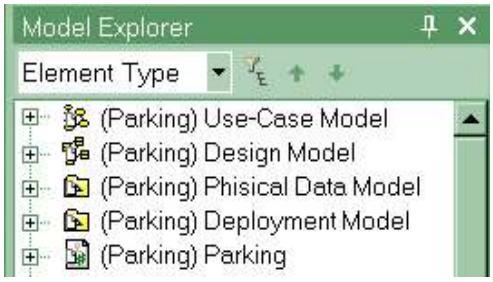
Istotną cechą struktury projektu w opisywanym środowisku jest podział repozytorium modeli na pakiety kontrolowane (w XDE reprezentowane przez pliki odpowiadające tworzonym modelom [11]) – minimalne, współużytkowane przez udziałowców elementy systemu stanowiące jego zasoby. Każdy z modeli stanowi samodzielny pakiet (kontrolowany), który jest powiązany z innymi modelami, zgodnie z semantyką języka UML.

Do budowy struktury projektu, zgodnej z metodyką Rational Unified Process [13], którą wspiera XDE, stosujemy następujące modele:

- model przypadków użycia (ang. *Use Case Model*),
- model analizy (ang. *Analysis Model*) – opcjonalny,
- model projektu (ang. *Design Model*),
- model implementacji (ang. *Implementation Model*),
- model wdrożenia (ang. *Deployment Model*),
- model kodu (ang. *Code Model*).

Natomiast w procesie modelowania bazy danych zastosowanie znajdują:

- model dziedziny (ang. *Domain Model*) – opcjonalny,
- logiczny model bazy danych (ang. *Logical Data Model*) – opcjonalny,
- fizyczny model danych (ang. *Physical Data Model*).

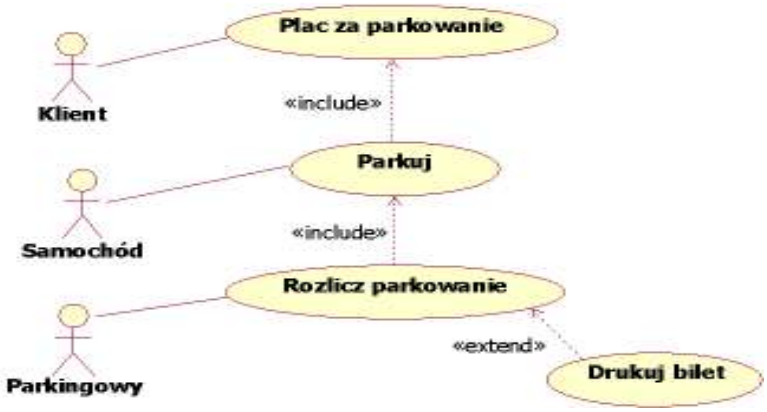


Rys. 6. Widok struktury projektu

Metodyka RUP pozwala na wybranie tych modeli, które w najlepszy sposób przedstawiają system. W przykładowej aplikacji webowej, budowanej na urządzenie przenośne, strukturę projektu tworzą: model przypadków użycia, model projektowy, fizyczny model bazy danych, model wdrożenia oraz model kodu.

6.1. Model przypadków użycia

Na podstawie wymagań funkcjonalnych zebranych w RequisitePro tworzy się diagramy przypadków użycia (ang. *Use-Case Diagrams*), które pozwalają na graficzne zaprezentowanie własności systemu, tak jak są one widziane po stronie użytkownika [12].



Rys. 7. Diagram przypadków użycia

Diagramy przypadków użycia są zbudowane z aktorów (ang. *actors*), czyli funkcji, które pełni użytkownik w stosunku do systemu oraz przypadków użycia (ang. *use-cases*), czyli zbioru scenariuszy powiązanych ze sobą wspólnym celem użytkownika [4].

Proces tworzenia modeli przypadków użycia, w opisywanym zintegrowanym środowisku programistycznym, przebiega podobnie dla każdego typu aplikacji, w tym i aplikacji webowych na systemy mobilne. W omawianym przykładzie diagram przypadków użycia (rys. 7) wskazuje na usługi realizowane przez system. Usługi te stanowią wizualizację wymagań funkcjonalnych aplikacji.

6.2. Model projektu

Opracowany model wymagań podlega analizie, a następnie, jako kolejne działania w cyklu wytwórczym, wykonywane są prace projektowe, które umożliwiają wizualizację sposobu działania tworzonego systemu. Model projektu, jako najważniejszy artefakt projektowy, zawiera decyzje wpływające na dalsze działania projektowe i ma decydujący wpływ na implementację systemu. Z tego też powodu, dla lepszego zrozumienia systemu, należy zastosować jego dekompozycję.

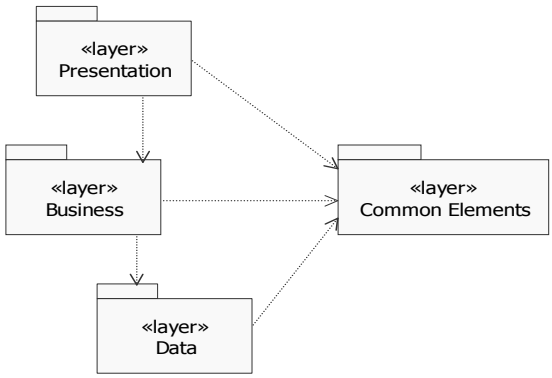
Dekompozycja aplikacji może nastąpić na kilka poziomów, z których każdy konceptualnie odpowiada za inne aspekty funkcjonowania systemu.

Należy zwrócić uwagę na to, że logiczne elementy warstw (poziomy dekompozycji) nie reprezentują ich fizycznego rozmieszczenia. Warstwy mogą znajdować się zarówno po stronie serwera, jak i po stronie klienta (tab. 2).

Tab. 2. Wybrane architektury aplikacji wielowarstwowych

Aplikacja		Warstwy	
		Strona klienta	Strona serwera
Terminalowa			prezentacja usługi dane
Klient-serwer	Cienki klient	prezentacja	usługi dane
	Gruby klient	prezentacja usługi	dane

Projektowany system jest aplikacją webową (implementowaną w technologii ASP.NET), dedykowaną dla urządzeń przenośnych. Z tego też powodu zdecydowano się na konfigurację tzw. „cienkiego klienta". Zasadniczo dokonuje się podziału na trzy warstwy, ale Rational XDE oferuje cztery warstwy (rys. 8). Każda z warstw jest agregatem komponentów o określonej funkcjonalności.



Rys. 8. Dekompozycja horyzontalna – podział na warstwy w XDE

Warstwa prezentacji (ang. *Presentation Layer*) jest odpowiedzialna za komunikację użytkownika z systemem. Znajdują się tu klasy interfejsu (ang. *Boundary Class*), które specyfikują strony aspx. Warstwa usług (ang. *Business Layer*) zawiera klasy sterujące (ang. *Control Class*), które sterują aplikacją i odpowiadają za jej działanie. Poziom ten stanowi most pomiędzy warstwami danych i prezentacji. Warstwa danych (ang. *Data Layer*) reprezentuje bazę danych i zawiera klasy danych (ang. *Entity Class*). Ostatnim elementem jest warstwa wspólna (ang. *Common Elements Layer*), która zawiera elementy, z których korzystają klasy zawarte w poprzednich trzech warstwach.

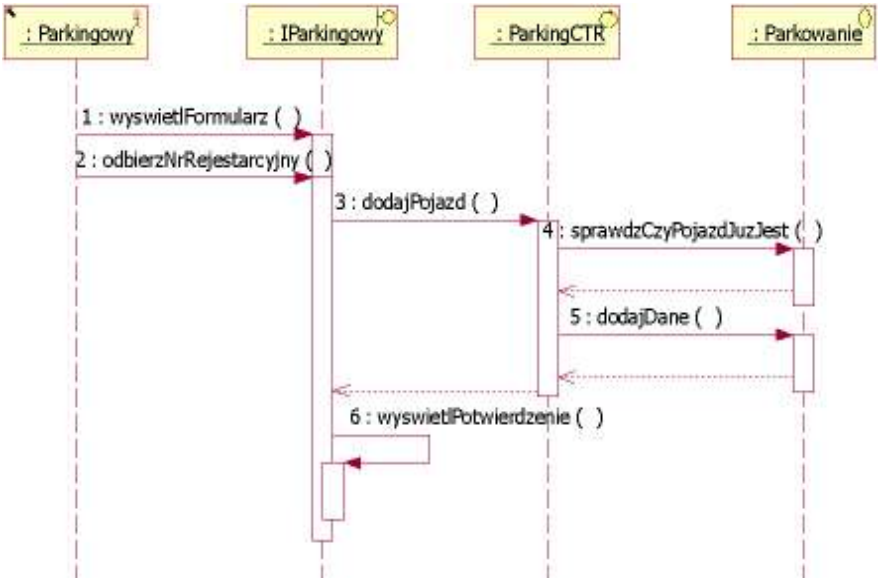
Istotną cechą tak zdefiniowanej architektury jest to, że komponenty z jednej warstwy mogą współpracować tylko z komponentami z warstwy bezpośrednio z nią sąsiadującej [2].

Cecha ta pozwala na wykorzystanie komponentów logiki biznesowej i warstw danych jako podstawy do zbudowania aplikacji webowych działających nie tylko na urządzeniach przenośnych, ale także w innych środowiskach. Z powyższych faktów wynika, że przy budowie aplikacji przeznaczonych na systemy mobilne, kluczowym elementem jest odpowiednia implementacja warstwy prezentacji, która pozwoli na odpowiednie wykorzystanie charakterystycznych cech urządzeń przenośnych.

Model projektowy, poza podzielonymi na warstwy klasami, zawiera uszczegółowienie przypadków użycia w postaci ich realizacji (ang. *Use Case Realizations*). Realizacje te przedstawiają uściślenia opisów statyki i dynamiki projektowanych przypadków użycia, określając, jak system będzie spełniał swoje obowiązki (wymagania). W aspekcie opisu statyki tworzone są diagramy klas, jako VOPC – *View of Participating Class*, natomiast aspekt dynamiki precyzują diagramy aktywności i sekwencji. Istotny jest fakt, że wszystkie diagramy budowane są przy użyciu zasady „przeciągnij i upuść" (ang. *drag&drop*), co przyspiesza budowę kolejnych diagramów.

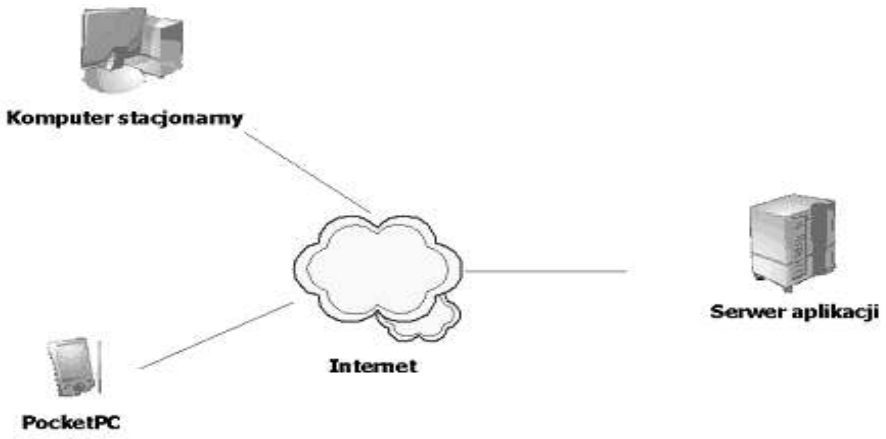
Rys. 9. Diagram klas (VOPC) przypadku użycia „Parkuj"

Na diagramie, przedstawionym na rysunku 9, zaprezentowano powiązania pomiędzy klasami: interfejsu – IParkingowy, sterowania – ParkingCTR i danych – Parkowanie. Diagram sekwencji (rys. 10) przedstawia natomiast scenariusz „Wjazd pojazdu na parking".



Rys. 10. Diagram sekwencji

W tym miejscu należy wspomnieć, że opisywane zintegrowane środowisko programistyczne nie wspiera projektantów w zakresie rozszerzeń języka UML dla systemów real-time'owych, ale pozwala na rozszerzenie zakresu stereotypów o dowolnie zdefiniowaną notację (rys. 11), w tym notację dedykowaną dla systemów RT.



Rys. 11. Diagram wdrożenia – przykład zastosowania nieformalnych stereotypów języka UML

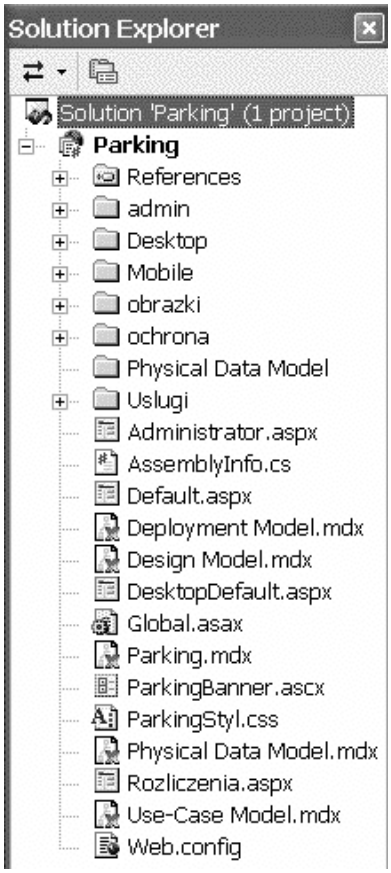
Z tego też powodu, aby móc skorzystać z elementów zawierających stereotypy preferowane w projektowaniu aplikacji czasu rzeczywistego, należy dodać je samodzielnie. Cecha ta jest bardzo istotna dla osób nieznających dobrze języka UML, gdyż wtedy mogą się one posługiwać mniej formalnymi piktogramami (obrazami stereotypów, dla których zdefiniowano składnię i semantykę) języka UML.

Możliwość budowania modeli na różnym poziomie abstrakcji, zastosowanie wielowarstwowej architektury projektu oraz przenoszenie modeli pomiędzy projektami stawia Rational XDE na pozycji wydajnego środowiska służącego do projektowania aplikacji webowych.

6.3. Implementacja

Kodowanie aplikacji w zintegrowanym środowisku programistycznym, jakie stanowi połączenie Visual Studio .NET i Rational XDE, opiera się głównie na wykorzystywaniu platformy .NET. Visual Studio .NET oferuje deweloperom

możliwość programowania w ponad czterdziestu językach, w tym: Visual Basic, Visual C++, C#, J# itd. Do wyboru jest także kilkanaście różnych szablonów, które pozwalają na budowanie wydajnych aplikacji okienkowych (desktopowych), webowych, terminalowych oraz mobilnych (na urządzenia przenośne, np. na platformie PocketPC). Wzorce pomagają tworzyć rozwiązania oparte m.in. na technologiach ASP.NET, XML i Web Services.



Rys. 12. Struktura repozytorium projektu

Zastosowanie wielowarstwowej struktury aplikacji umożliwiło rozgraniczenie odpowiedzialności zbudowanych klas (rys. 12) w ten sposób, że kod klas odpowiedzialnych za obsługę interfejsu użytkownika znajduje się w plikach posiadających rozszerzenie .aspx. Klasy te zostały zbudowane na podstawie klas zamieszczonych w warstwie prezentacji modelu projektowego. Klasy z zaprojektowanej warstwy usług (model projektu) zostały zaimplementowane w plikach z rozszerzeniem .cs, które znajdują się w pakiecie Usługi.

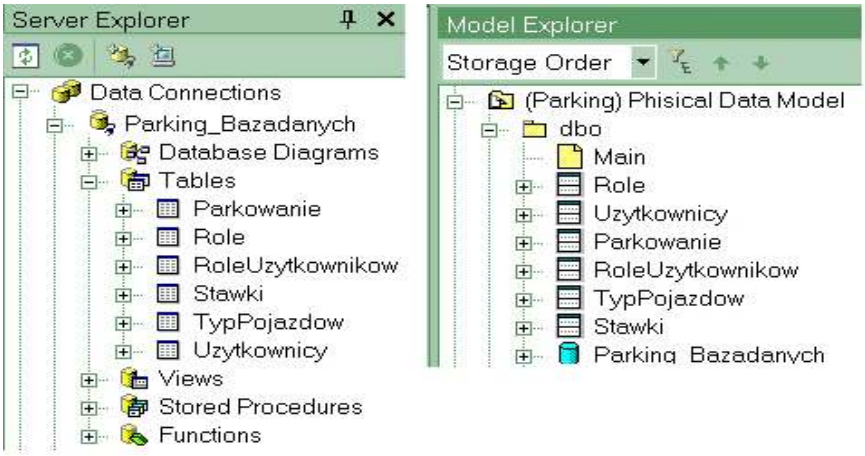
7. Synchronizacja

Zintegrowanie Visual Studio .NET z Rational XDE w jedno środowisko programistyczne umożliwiło, w każdym momencie procesu wytwórczego, synchronizację modeli z ich implementacją zarówno na poziomie warstwy prezentacji, usług, jak i warstwy danych.

7.1. Synchronizacja modelu z bazą danych

Zastosowanie zintegrowanego narzędzia wytwórczego, opartego na VS.NET i XDE, pozwala na wykorzystywanie repozytorium projektu modelu danych, który jest w pełni zsynchronizowany zarówno z kodem, jak i z fizycznie istniejącą bazą danych. W łatwy sposób można przenieść projekt bazy danych (jako metamodel) do konkretnego środowiska implementacji, którym może być przykładowo Microsoft SQL Server, Oracle, a za pomocą mechanizmów ODBC – praktycznie dowolnie wybrana baza danych.

W prezentowanym przykładzie zastosowanie metod inżynierii w przód (ang. *forward engineering*) pozwoliło na wygenerowanie, na podstawie modelu fizycznej bazy danych, odpowiedniej struktury tabel na serwerze bazodanowym (rys. 13). Dodatkowo, za pomocą inżynierii wstecz (ang. *reverse engineering*) i synchronizacji, przy użyciu odpowiednich kreatorów, opisywane środowisko umożliwia uzyskanie zgodności pomiędzy modelem bazy danych z jego implementacją [5] i [9].



Rys. 13. Widoki bazy danych (Server Explorer) i modelu danych (Model Explorer)

7.2. Synchronizacja modelu z jego implementacją

Wzajemna współpraca XDE z VS.NET zaowocowała możliwością pełnej synchronizacji kodu źródłowego aplikacji z modelem.

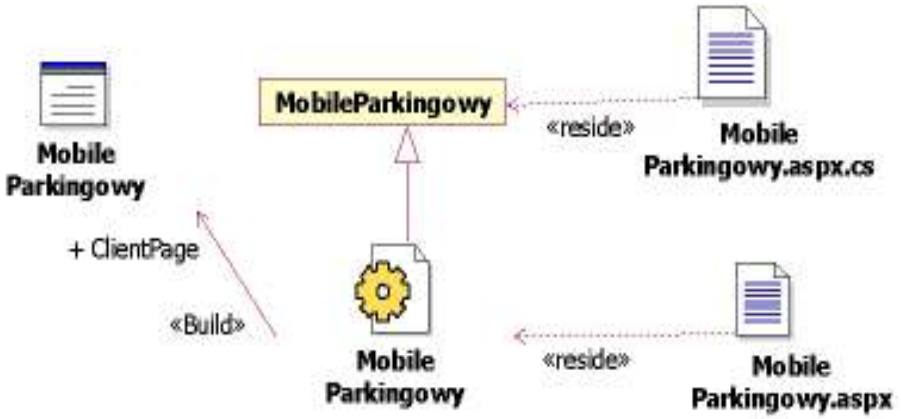
Wynikiem synchronizacji jest zatem kod zgodny z modelem i odwrotnie (rys. 14). Model kodu zawiera trzy foldery. W pakiecie Artifacts znajdują się pliki i pakiety, które są artefaktami wytworzonymi w procesie implementacji

aplikacji. W plikach tych znajdują się klasy odpowiedzialne za interfejsy i usługi aplikacji. Wszystkie klasy składające się na budowaną aplikację znajdują się w przestrzeni nazw Parking, która została określona poprzez pakiet Parking. Ostatnim folderem występującym modelu kodu jest pakiet References, który zawiera powiązania z innymi przestrzeniami nazw. Jak już wspomniano, synchronizację kodu z modelem można wykonać na dowolnym etapie wytwarzania aplikacji. Dodatkowo, można tak skonfigurować narzędzie, by samo dokonywało automatycznej synchronizacji, która może mieć miejsce po każdej zmianie czy to w kodzie programu, czy to w modelu.

Rys. 14. Fragment modelu kodu

Proces synchronizacji, dzięki inżynierii wstecz, pozwala na utworzenie modelu kodu z wykorzystaniem stereotypów języka UML, dedykowanych dla wybranych typów aplikacji, w tym i aplikacji webowej.

W omawianym przykładzie, w zdefiniowanej przestrzeni nazw znajdują się nie tylko klasy. Dzięki stereotypom języka UML wyszczególnione są bowiem także inne elementy wchodzące w skład budowanego systemu, takie jak strona klienta czy strona serwera. Diagram kodu (rys. 15) prezentuje zależności pomiędzy elementami wchodzącymi w skład modelu kodu.



Rys. 15. Przykładowy diagram kodu systemu „Parkingowy"

8. Badanie własności dynamicznych systemu

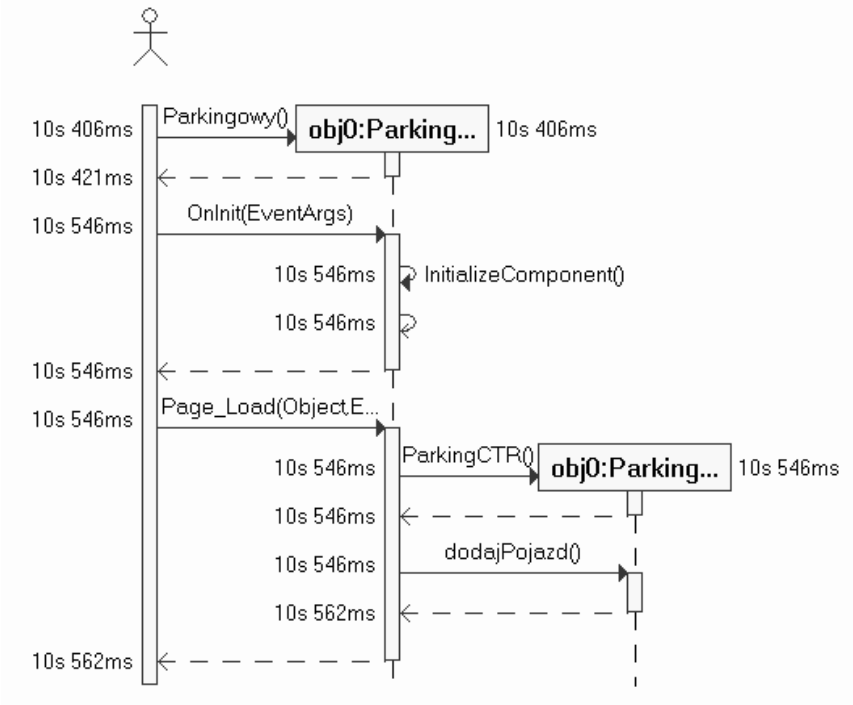
Opisywane zintegrowane środowisko programistyczne pozwala, dzięki narzędziom inżynierii w przód, wygenerować szkielet aplikacji i zbadać jego własności dynamiczne. Oznacza to, że Visual Studio i XDE umożliwiają badanie własności dynamicznych systemu już na poziomie specyfikacji wyrażonej w języku UML, w całym procesie budowania systemu.

Do badania cech behawioralnych aplikacji wykorzystywane są, wchodzące w skład Rational XDE narzędzia, takie jak: Visual Trace i Quantify. Dodatkowo, wsparcie dla wyszukiwania błędów w kodzie stanowi narzędzie Purify, a pokrycie kodu testami możemy analizować dzięki PureCoverage.

8.1. Automatyczne tworzenie diagramów sekwencji

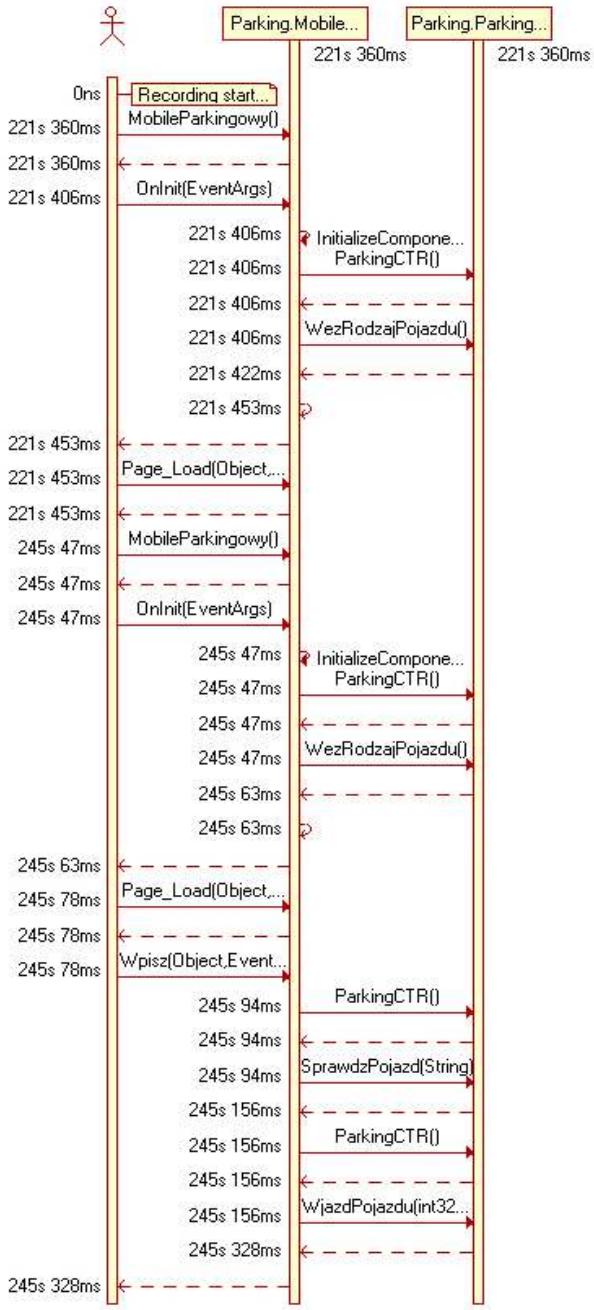
Opisywane zintegrowane środowisko wytwórcze posiada bardzo interesującą cechę, a mianowicie pozwala na wygenerowanie śladowych diagramów sekwencji (ang. *trace sequence diagram*) [10], wywodzących się z diagramów sekwencji języka UML. Diagramy te powstają na podstawie uruchomienia implementacji (kodu wynikowego i statycznych cech implementacji opisanych w kodzie źródłowym) bezpośrednio w trakcie jej działania. Możliwości te są dostępne dzięki zastosowaniu narzędzia Visual Trace, które jest integralnym składnikiem IBM Rational Developer Plus. Visual Trace pozwala na wygenerowanie diagramów śladowych zarówno z aplikacji

Windows, jak i webowych. Zbudowany w przez Visual Trace śladowy diagram sekwencji zawiera typowe elementy znane z UML, takie jak: obiekty, linie życia i komunikaty. Diagram taki można uzyskać już z poziomu obiektów projektowych budowanego systemu – jako wynik uruchomienia jego specyfikacji (rys. 16).



Rys. 16. Śladowy diagram sekwencji wykonany z poziomu specyfikacji systemu – „Wjazd pojazdu na parking"

Na przedstawionych przykładach (rys. 16 i 17) można zobaczyć, że czas każdej czynności został zmierzony z dokładnością do tysięcznych części sekundy. Jest to cecha bardzo istotna przy testowaniu i tworzeniu dokumentacji systemów czasu rzeczywistego.



Rys. 17. Śladowy diagram sekwencji – „Wjazd pojazdu na parking”

Porównując zaprezentowane trzy diagramy sekwencji: diagram sekwencji wytworzony w trakcie projektowania systemu (rys. 10) ze śladowymi diagramami sekwencji (rys. 16 i 17) można stwierdzić, że choć wszystkie wymienione diagramy opisują ten sam scenariusz (Wjazd pojazdu na parking), to uwypuklają aspekt behawioralny systemu na różnych poziomach abstrakcji. Diagram sekwencji (rys. 10) przedstawia model konceptualny scenariusza głównego usługi „Wjazd pojazdu na parking". Śladowe diagramy sekwencji ilustrują natomiast w jaki sposób działa system.

Należy jednak zwrócić uwagę na fakt, że przedstawiony śladowy diagram sekwencji (rys. 16) pozwala na badanie własności behawioralnych aplikacji już w trakcie projektowania systemu. Kolejny zaprezentowany diagram sekwencji (rys. 17) przedstawia natomiast w pełni zaimplementowaną funkcjonalność systemu.

Taka własność środowiska umożliwia badanie aplikacji już w trakcie projektowania, co w konsekwencji pozwala na wczesne wprowadzanie zmian w projektowanym komponencie, gdy nie spełnia on postawionych wcześniej wymagań czasowych.

Budowanie diagramów sekwencji za pomocą Visual Trace jest doskonałą metodą na uzupełnienie kodu i jego modelu o automatycznie uzyskiwane dynamiczne aspekty budowanego systemu w postaci śladowego diagramu sekwencji wykorzystującego standardową notację języka UML.

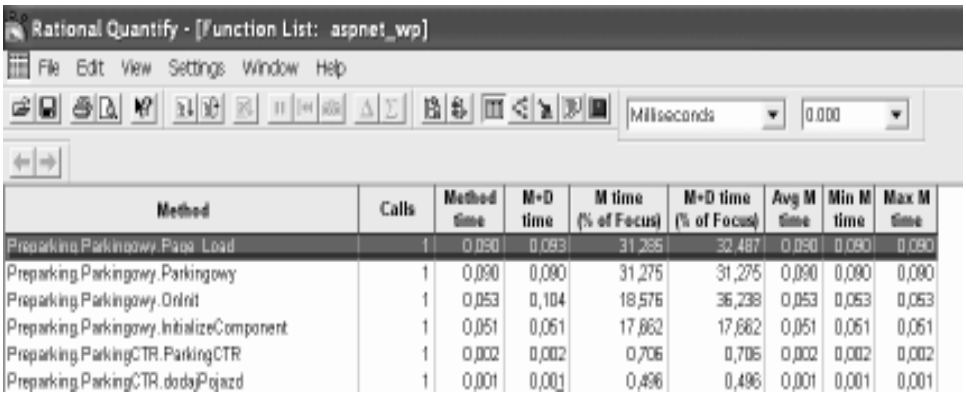
Istotny jest także fakt, że wytworzony śladowy diagram sekwencji za pomocą odpowiednich filtrów może być prezentowany z wybraną szczegółowością, a po skończonych badaniach może zostać zarchiwizowany w istniejącym repozytorium projektu.

8.2. Optymalizacja wydajności kodu aplikacji

Do badania behawioralnych aspektów systemu, poza wspomnianym wcześniej Visual Trace, można wykorzystać inne narzędzie, wchodzące w skład Rational XDE, a mianowicie Quantify.

Rational Quantify jest zaawansowanym narzędziem służącym do optymalizacji wydajności aplikacji, które wskazuje tzw. wąskie gardła systemu. Dzięki graficznej metodzie prezentacji danych, informującej o wydajności, projektant z dużą łatwością, na podstawie wyników działania aplikacji jest w stanie wykryć te funkcje, których działanie nie spełnia postawionych wcześniej wymagań.

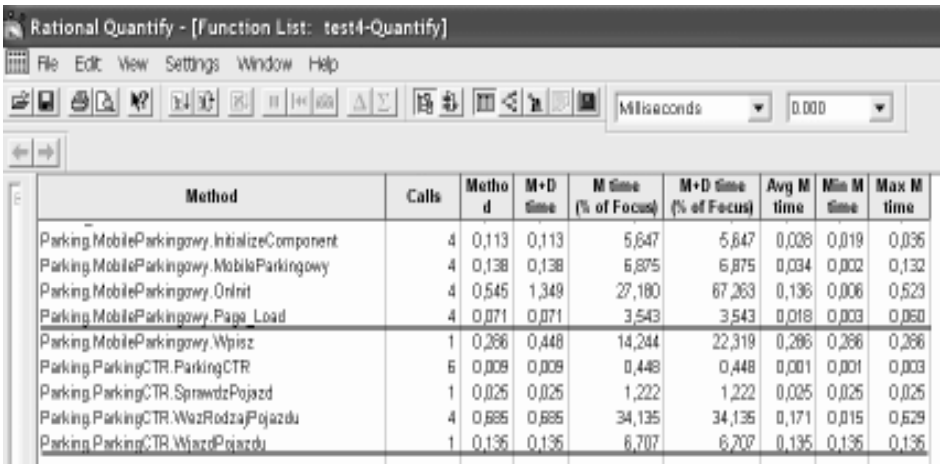
Quantify umożliwia, dzięki inżynierii w przód dostępnej w XDE, wykonanie badań zachowania systemu już z poziomu specyfikacji systemu wyrażonej w języku UML (rys. 18).



Method	Calls	Method time	M+D time	M time (% of Focus)	M+D time (% of Focus)	Avg M time	Min M time	Max M time
Preparking.Parkingowy.Page_Load	1	0,090	0,093	31,286	32,487	0,090	0,090	0,090
Preparking.Parkingowy.Parkingowy	1	0,090	0,090	31,275	31,275	0,090	0,090	0,090
Preparking.Parkingowy.OnInit	1	0,053	0,104	18,576	36,238	0,053	0,053	0,053
Preparking.Parkingowy.InitializeComponent	1	0,051	0,051	17,662	17,662	0,051	0,051	0,051
Preparking.ParkingCTR.ParkingCTR	1	0,002	0,002	0,706	0,706	0,002	0,002	0,002
Preparking.ParkingCTR.dodajPojazd	1	0,001	0,001	0,496	0,496	0,001	0,001	0,001

Rys. 18. Quantify – wyniki testu wydajnościowego poziomu specyfikacji systemu

Testy wydajnościowe aplikacji można wykonywać przez cały czas projektowania i implementacji aplikacji aż po gotowy produkt (rys. 19), tak by otrzymany wynik był zgodny z wymaganiami stawianymi poszczególnym komponentom.



Method	Calls	Method	M+D time	M time (% of Focus)	M+D time (% of Focus)	Avg M time	Min M time	Max M time
Parking.MobileParkingowy.InitializeComponent	4	0,113	0,113	5,647	5,647	0,028	0,019	0,035
Parking.MobileParkingowy.MobileParkingowy	4	0,138	0,138	6,875	6,875	0,034	0,002	0,132
Parking.MobileParkingowy.OnInit	4	0,545	1,349	27,180	67,263	0,136	0,006	0,523
Parking.MobileParkingowy.Page_Load	4	0,071	0,071	3,543	3,543	0,018	0,003	0,060
Parking.MobileParkingowy.Wpisz	1	0,286	0,448	14,244	22,319	0,286	0,286	0,286
Parking.ParkingCTR.ParkingCTR	6	0,009	0,009	0,448	0,448	0,001	0,001	0,003
Parking.ParkingCTR.SprawdzPojazd	1	0,025	0,025	1,222	1,222	0,025	0,025	0,025
Parking.ParkingCTR.WazRodzajPojazdu	4	0,685	0,685	34,135	34,135	0,171	0,015	0,629
Parking.ParkingCTR.WjazdPojazdu	1	0,135	0,135	6,707	6,707	0,135	0,135	0,135

Rys. 19. Quantify – wyniki testu wydajnościowego w pełni zaimplementowanego systemu

Na przedstawionych rysunkach (rys. 18 i 19) można zauważyć, że Quantify umożliwia badanie liczby odwołań do metod oraz czasów ich wykonywania: średnich, minimalnych i maksymalnych. Wszystkie badania mogą być wykonywane dla odpowiednio wcześniej zdefiniowanego filtru, który kolekcjonuje zdarzenia dotyczące badanych metod (na właściwym poziomie abstrakcji – od obiektów projektowych, poprzez kod źródłowy do ich kodu asemblerowego).

8.3. Podsumowanie badania własności behawioralnych systemu

Przedstawione narzędzia Visual Trace i Quantify ułatwiają badanie własności dynamicznych systemu już od poziomu specyfikacji systemu wyrażonej w języku UML aż po gotową aplikację.

Należy jednak zwrócić uwagę na fakt, że dane uzyskane na podstawie śladowych diagramów sekwencji pozwalają jedynie na porównanie czasów wykonywania metod (uzyskane wyniki stanowią łączny czas wykonania metody i „rysowania" diagramu, co znacznie zniekształca uzyskane pomiary), w odróżnieniu od precyzyjnych danych uzyskanych z Quantify. Porównanie wyników otrzymanych pomiarów przedstawiono w tabeli 3.

Tab. 3. Wyniki pomiarów czasów wykonania wybranych metod systemu

Rodzaj badania	Nazwa metody	Visual Trace [ms]	Quantify [ms]
Szkielet projektu	Parkingowy.Page_Load	16	0,090
	ParkingCTR.dodajPojazd	16	0,001
Gotowa aplikacja	MobileParkingowy.Page_Load	283	0,071
	ParkingCRT.WjazdPojazdu	63	0,135

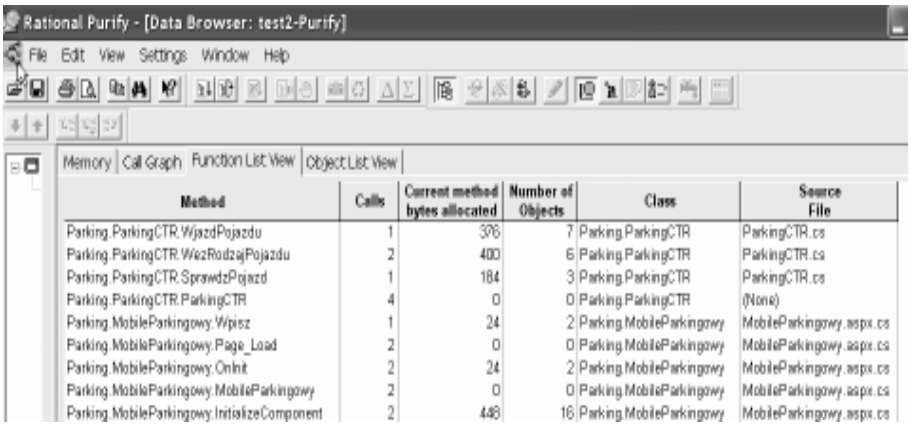
9. Możliwości zintegrowanego środowiska

Prezentowane zintegrowane środowisko programistyczne, oprócz opisanych wcześniej własności, umożliwia także badanie aplikacji w innych aspektach niż wymienione oraz pozwala deweloperom na tworzenie repozytoriów wielokrotnego użycia.

9.1. Badania dodatkowe

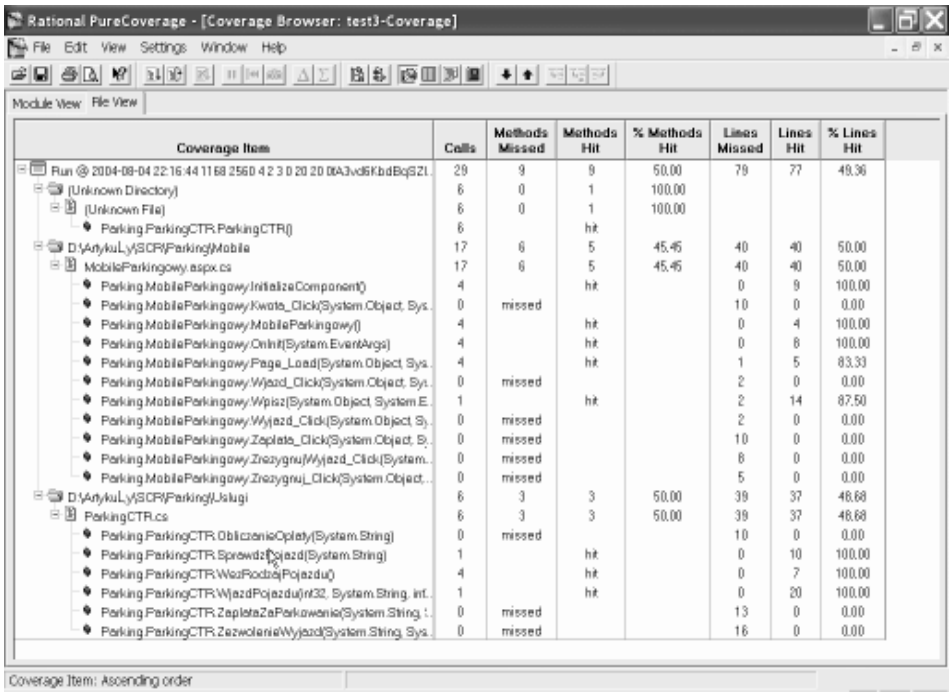
Opisywane zintegrowane środowisko pozwala, poza testami wydajnościowymi oferowanymi przez Quantify, na uzyskanie testów wykonania i analizę pokrycia kodu testami.

Do wyszukiwania błędów kodu oraz błędów pamięci służy Rational Purify, który pozwala na wyświetlenie informacji o znalezionych błędach.



Rys. 20. Purify – wynik poszukiwań błędów wykonania i pamięci

Drugim narzędziem wspierającym zespół projektowy jest Rational PureCoverage, który pozwala na analizę pokrycia kodu testami.



Rys. 21. PureCoverage – raport pokrycia kodu testami

Istotny jest fakt, że testy, wykonane zarówno przy pomocy Purify, jak i Quantify, mogą być realizowane już na poziomie obiektów projektowych, wyrażonych w UML. Wszystkie testy mogą być zrealizowane dla zdefiniowanych filtrów, które kolekcjonują zdarzenia o wybranych metodach.

9.2. Repozytorium artefaktów wielokrotnego użycia

Opisywane zintegrowane środowisko posiada jeszcze jedną ważną cechę. Umożliwia deweloperom tworzenie repozytoriów wielokrotnego użycia, z których może skorzystać każdy z członków zespołu projektowego. Repozytoria te, zwane *Resuable Asset Specification* (RAS), wspomagają proces wytwórczy oprogramowania, gdyż pozwalają za pomocą odpowiedniego kreatora wyeksportować i zaimportować wybrane elementy projektu. Właściwość ta jest bardzo ciekawa, zwłaszcza przy wytwarzaniu „podobnych" systemów. Wówczas zaimportowanie fragmentu modelu i kodu programu pozwala na szybkie dodawanie kolejnych komponentów systemu. RAS jest tak skonstruowane, że umożliwia opublikowanie w sieci kolejnych wersji, tak by wszyscy udziałowcy projektu mieli do nich dostęp. Dodatkowo, zarządzanie poszczególnymi repozytoriami jest wsparte przez odpowiednią przeglądarkę, co zostało przedstawione na rysunku 22.

Rys. 22. Repozytorium RAS

10. Zakończenie

Przedstawione zintegrowane środowisko programistyczne pozwala na utrzymanie spójności między artefaktami projektu w całym procesie wytwórczym aplikacji. Łączne zastosowanie XDE i VS.NET umożliwia utrzymywanie w jednym repozytorium systemu modeli o różnym poziomie abstrakcji – od konceptualnego (wyobrażeniowego) modelu usług (XDE), wyrażonego na diagramach przypadków użycia, po model prezentujący kod

aplikacji – model implementacyjny, do kodu źródłowego (VS.NET). Dodatkowo, synchronizacja kodu programu z jego modelem ułatwia zarządzanie zmianami w projekcie, tworząc trwałą relację między nimi (tzw. „żywy kod"). Zastosowanie Visual Trace oraz innych wymienionych w artykule narzędzi, pozwala natomiast na uzyskanie wizualnej prezentacji cech behawioralnych działającego kodu systemu. Dodatkowo, narzędzia te dają możliwość śledzenia rozwiązania na „poziomie obiektów projektowych", co sprawia, że są one pożądane do badania i dokumentowania aspektów dynamicznych tworzonych systemów.

Ważny jest fakt, że wszystkie testy zachowania systemu można wykonać dla wybranych klas systemu, co powala na skupienie uwagi zespołu projektowego na wybranych aspektach aplikacji. Badanie aspektów dynamicznych systemu reaktywnego już od powstania jego specyfikacji umożliwia, w przypadku niespełnienia postawionych wymagań, jej wczesne przeprojektowanie, a co za tym idzie – zmniejszenie kosztów związanych z poprawianiem gotowego produktu.

Powyższe cechy, łącznie z możliwością tworzenia repozytoriów wielokrotnego użycia, skłaniają do stwierdzenia, że zintegrowane środowisko programistyczne jest fabryką oprogramowania, która stanowić może alternatywę dla oprogramowania dedykowanego dla tej klasy rozwiązań (np. narzędzia – Rational Suite Development Studio – RealTime Edition). W świetle przedstawianych cech poprawne wydaje się również twierdzenie, że skutecznym sposobem budowania aplikacji mobilnych może być wykorzystanie zintegrowanego środowiska XDE z VS.NET.

Obecnie prezentowane środowisko zasadniczo ukierunkowane jest na wspieranie Rational Unified Process. Dzięki modelom opartym o pakiety kontrolowane wspierać może coraz bardziej popularne *Model Driven Architecture* (MDA). Wydaje się jednak, że jest to tylko kolejny etap rozwoju narzędzi deweloperskich, które zmierzają do w pełni zintegrowanej platformy programistycznej, zoptymalizowanej pod kątem roli każdego członka zespołu projektowego oraz etapu procesu wytwórczego, w którym dane narzędzie jest stosowane. Z tego też powodu, coraz częściej, środowiska wytwórcze oprogramowania odwoływać się będą do języków modelowania (np. UML) i koncepcji procesu wytwórczego wspierającego Model-Driven Development (MDD).

Literatura

[1] Booch G., Rumbaugh J., Jacobson I..: *UML przewodnik użytkownika*, WNT, Warszawa, 2003.

[2] Calvez J. P.: *Embedded Real-Time Systems.* John Wiley & Sons, 1993.

[3] Eeles P.: *Layering Strategies*, Rational Software, 2001.

[4] Flower M., Kendall S.: *UML w kropelce*, Oficyna Wydawnicza LTP, Warszawa, 2002.

[5] Franklin S.: *Data Modeling in Rational XDE Release 2*, www.rational.net, 2003.

[6] Goldsmith S.: *A Practical Guide to Real-Time Systems Development*. Prentice Hall Int., 1993.

[7] IBM *Rational XDE Developer v2003.06.12* – .NET Edition Evaluators Guide.

[8] Kroll P., Kruchten P.: *Rational Unified Process Made Easy*: A Practitioner's Guide to the RUP, Addison Wesley, 2003.

[9] Kuslich J.: *Data Modeling in Rational XDE Release 2*: A Guide for Visual Studio .NET Developers, www.rational.net, 2003.

[10] Kuslish J.: *Visual Trace and Asset Repositories in Rational XDE Developer v2003*: A Guide for Visual Studio .NET Developers, www.rational.net, 2003.

[11] *Model Structure Guidelines for Rational XDE Developer* - .NET Edition, Rational Unified Process, wersja 2003.06.12.

[12] Mrozek Z.: *Metodyka wykorzystania UML w projektowaniu mechatronicznym,* Pomiary Automatyka Kontrola, PAK 1/2002.

[13] *Rational Unified Process*, wersja 2003.06.12.

[14] Stasiak A., Wolski M.: *Zintegrowane środowisko wytwarzania aplikacji webowych dla systemów mobilnych na platformie .NET*, XI Konferencja SCR'04, Ustroń 13-16 września 2004 r., 243-252.

[15] Stasiak A., Zieliński Z.: *Analiza i projektowanie systemów wbudowanych z wykorzystaniem Rational Unified Process (RUP)*, Biuletyn IAiR, nr 16, WAT, Warszawa, 2001.

[16] Yourdon E.: *Marsz ku klęsce – poradnik dla projektanta systemów*, WNT, Warszawa, 2000.

Recenzent: prof. dr hab. inż. Włodzimierz Kwiatkowski

Praca wpłynęła do redakcji: 27.12.2005