
Helion
DDD
KOMPENDIUM
WIEDZY
VAUGHN VERNON

Tytuł oryginału: Domain-Driven Design Distilled

Tłumaczenie: Maksymilian Gutowski

ISBN: 978-83-283-4279-8

Authorized translation from the English language edition, entitled: DOMAIN-DRIVEN DESIGN DISTILLED; ISBN 0134434420; by Vaughn Vernon; published by Pearson Education, Inc, publishing as Addison-Wesley Professional.

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

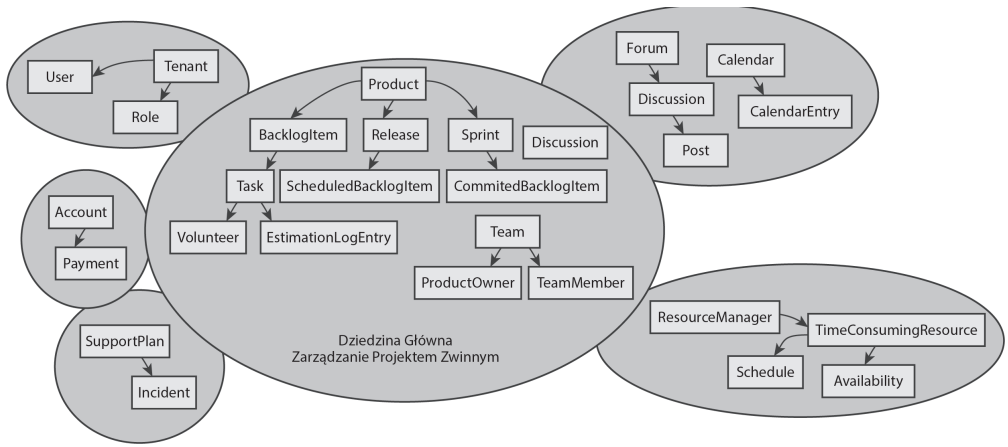
Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/dddpig
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

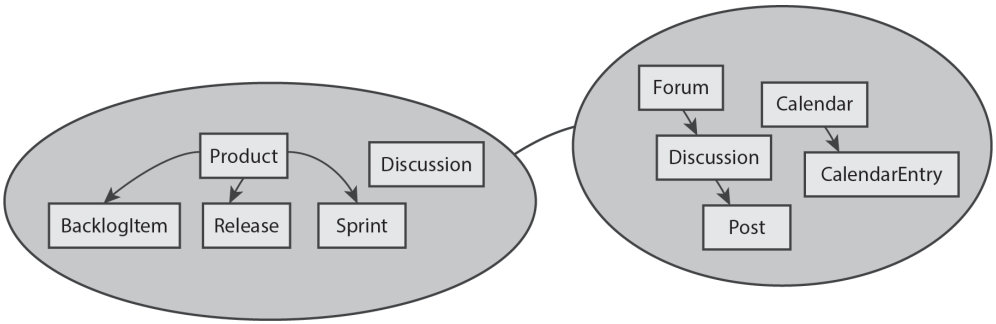
Spis treści

Rozdział 4

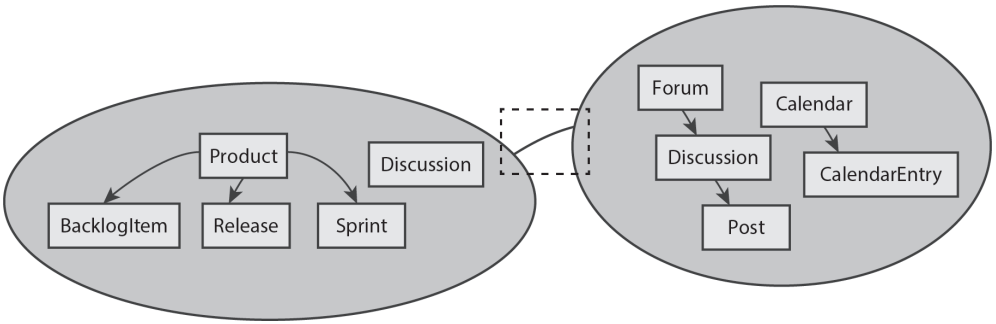
Projektowanie strategiczne z wykorzystaniem mapowania kontekstów



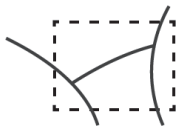
W poprzednich rozdziałach dowiedziałeś się, że w każdym projekcie DDD poza Dziedziną Główną istnieje wiele Kontekstów Ograniczonych. Wszystkie pojęcia, które nie należały do Kontekstu Zarządzanie Projektem Zwinnym — czyli Dziedziny Głównej — zostały przeniesione do osobnych Kontekstów Ograniczonych.



Dowiedziałeś się też, że Dziedzina Główna „Zarządzanie Projektem Zwinnym" powinna zostać zintegrowana z innymi Kontekstami Ograniczonymi. Taka integracja w DDD nosi nazwę Mapowanie Kontekstu. Na powyższej Mapie Kontekstu widać, że dyskusja (`Discussion`) istnieje w obydwu Kontekstach Ograniczonych. Jak pamiętasz, wynika to z tego, że Kontekst Współpraca jest źródłem komponentu `Discussion`, a Kontekst Zarządzanie Projektem Zwinnym jego konsumentem.



Mapowanie Kontekstu jest oznaczone w powyższym diagramie jako linia otoczona przerywaną obwódką. (Sama obwódka nie jest częścią Mapy Kontekstu, lecz jedynie ma zwrócić uwagę na linię). To właśnie linia łącząca obydwa Konteksty Ograniczone wskazuje na Mapowanie Kontekstu. Innymi słowy, obecność takiej linii świadczy o tym, że Konteksty Ograniczone są ujęte na jakiejś mapie. W ramach obydwu Kontekstów Ograniczonych zachodzi pewien stopień integracji, jak i interakcji pomiędzy odpowiadającymi im zespołami.



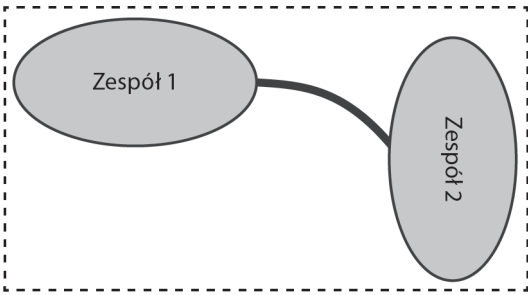
Biorąc pod uwagę, że w dwóch różnych Kontekstach Ograniczonych istnieją dwa różne Języki Wszechobecne, powyższa linia wskazuje, że występujące w nich pojęcia ulegają przetłumaczeniu. Wyobraź sobie, że dwa zespoły z dwóch różnych krajów, posługujące się dwoma różnymi językami, muszą ze sobą współpracować. Żeby mogło się to powieść, to albo obydwa zespoły musiałyby się nauczyć języka obcego, albo trzeba by skorzystać z usług tłumacza.

Zatrudnienie tłumacza odciążyłoby obydwa zespoły, ale jednocześnie mogłoby się okazać pod wieloma względami kosztowne. Zastanów się tylko, ile czasu pochłaniałoby przekazywanie informacji między jednym zespołem a drugim za pośrednictwem tłumacza. Obsługa tłumacza początkowo wydaje się wygodna, ale z czasem staje się męcząca. Tym niemniej zespoły mogą uznać, że jest to lepsze rozwiązanie w porównaniu z koniecznością nauczenia się nowego języka i ciągłego żonglowania językami. Rzecz jasna, właśnie opisaliśmy interakcję dwóch zespołów. Co, jeśli tych zespołów byłoby więcej? Z takimi samymi problemami mierzymy się, gdy tłumaczymy Języki Wszechobecne lub uczymy się nimi posługiwać.

W ramach Mapowania Kontekstu interesuje nas to, *jaką relację* pomiędzy zespołami oraz integracją opisuje linia łącząca dwa Konteksty Ograniczone. Odpowiednie zdefiniowanie dzielących je granic i kontraktów zachodzących pomiędzy nimi pozwala na wprowadzanie kontrolowanych zmian. Mapowanie Kontekstów przyjmuje różne formy, zarówno pod względem technicznym, jak i zespołowym. W niektórych przypadkach relacja między zespołami przeplata się z integracją.

Formy Mapowania Kontekstu

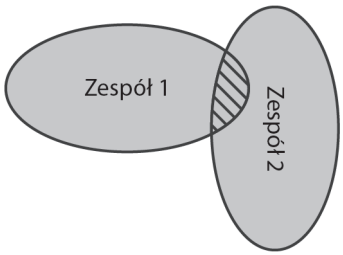
Jakie zależności oraz integracje w Mapowaniu Kontekstu można wyrazić za pomocą linii? Przedstawię Ci je poniżej.



Partnerstwo

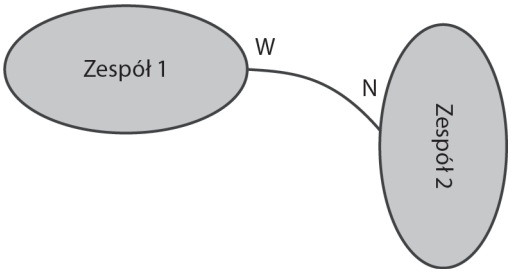
Pomiędzy zespołami zachodzi relacja Partnerstwa. Każdy zespół odpowiada za jeden Kontekst Ograniczony. Wspólnie tworzą one Partnerstwo, w ramach którego posługują się zależnymi zestawami celów. Oznacza to, że obydwa zespoły wspólnie odnoszą sukces lub porażkę. Ponieważ są one tak ściśle związane, członkowie zespołów będą się często spotykać w celu synchronizacji harmonogramów i prac, a ponadto będą stosować system ciągłej integracji, aby zachować spójność wyników. Synchronizację przedstawia się w formie grubej linii rozciągniętej pomiędzy dwoma zespołami. Gruba linia wskazuje na wymagany poziom zaangażowania, który jest w tym wypadku dość wysoki.

Utrzymanie Partnerstwa przez dłuższy czas jest nie lada wyzwaniem, wobec czego zespoły wkraczające w taką relację mogą zgodzić się na jej ograniczenie czasowe. Partnerstwo powinno trwać tak długo, jak długo dostarcza korzyści, a powinno zostać przekształcone w inną relację, gdy korzyści przestają być warte zaangażowania.



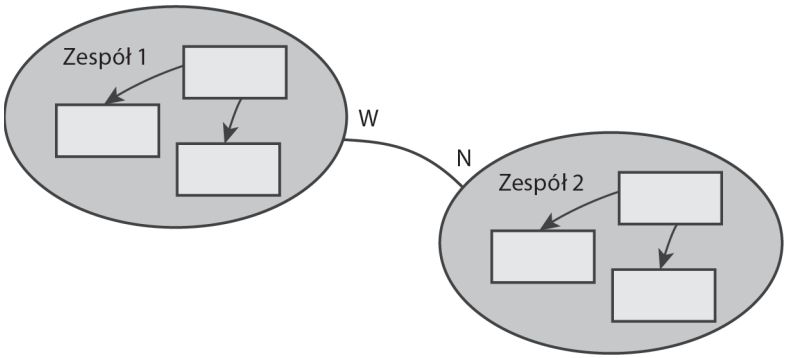
Wspólne Jądro

Wspólne Jądro, przedstawione powyżej jako część wspólna dwóch Kontekstów Ograniczonych, opisuje relację pomiędzy dwoma (lub więcej) zespołami, które współdzielą niewielki model. Zespoły muszą być zgodne co do tego, które elementy modelu mają być wspólne. Możliwe jest, że tylko jeden zespół będzie odpowiadał za utrzymanie kodu, wydawanie i testowanie współdzielonych zasobów. Wspólne Jądro często bardzo trudno w ogóle zdefiniować i utrzymać, ponieważ wiąże się to z koniecznością prowadzenia otwartej komunikacji międzyzespołowej i potwierdzania zgody co do tego, jakie elementy składają się na wspólny model. Niemniej może się to udać, o ile wszyscy zainteresowani są zgodni co do tego, że zachowanie takiego jądra jest lepsze niż Oddzielne Drogi (które omówię w dalszej części podrozdziału).



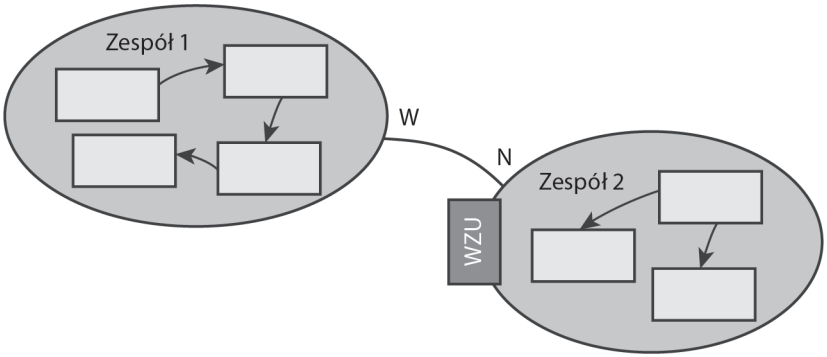
Rozwój w trybie Klient-Dostawca

Model Klient-Dostawca opisuje taką relację pomiędzy dwoma Kontekstami Ograniczonymi i ich zespołami, w ramach której Dostawca znajduje się na wyższym szczeblu (W na diagramie), Klient na niższym (N na diagramie). Dostawca dominuje w tej relacji, ponieważ odpowiada za dostarczanie tego, czego Klient potrzebuje. To do Klienta należy zaplanowanie różnych transakcji z Dostawcą, ale ostatecznie to ten pierwszy decyduje, co i kiedy trafi do Klienta. Jest to niezwykle standardowa i bardzo praktyczna relacja między zespołami, nawet należącymi do jednej organizacji, o ile kultura firmowa nie zapewnia Dostawcy całkowitej autonomii i możliwości lekceważenia realnych oczekiwań Klientów.



Konformista

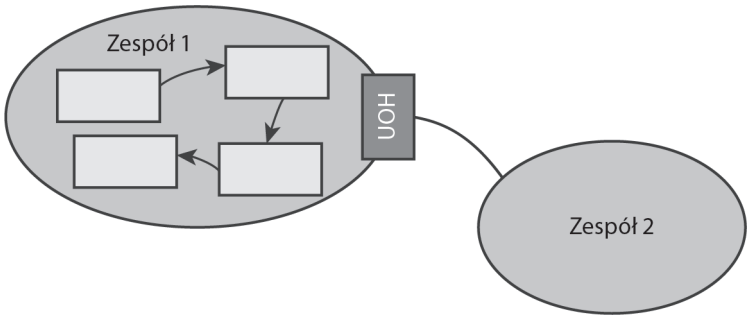
Relacja Konformisty zachodzi, kiedy istnieją zespoły na wyższym i niższym szczeblu, a zespół wyższego szczebla nie ma żadnej motywacji do zaspokajania potrzeb tego drugiego. Zespół niższego szczebla z różnych względów nie może pozwolić sobie na tłumaczenie Języka Wszechobecnego zespołu wyższego szczebla, więc dostosowuje się on do jego modelu. Zespoły często decydują się na bycie Konformistami, na przykład podczas integracji z bardzo dużym i skomplikowanym modelem, na który zespół nie może bezpośrednio wpływać. Przykładem może być konieczność dostosowania się do modelu Amazon.com w ramach integracji w charakterze stowarzyszonego sprzedawcy Amazona.



Warstwa Zapobiegająca Uszkodzeniu

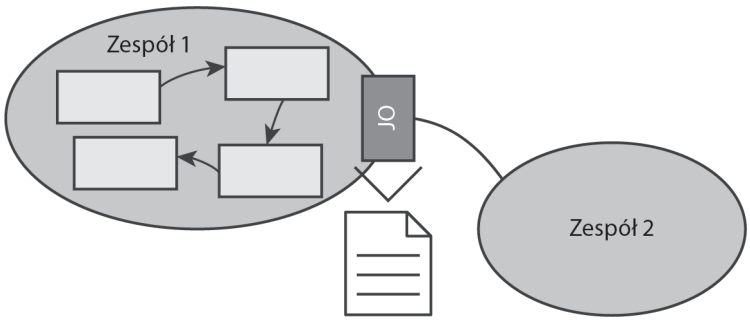
Warstwa Zapobiegająca Uszkodzeniu jest najbezpieczniejszą relacją, w ramach której zespół niższego szczebla tworzy warstwę translacyjną pomiędzy własnym Językiem Wszechobecnym a Językiem Wszechobecnym zespołu wyższego szczebla. Warstwa ta izoluje model niższego szczebla od jego odpowiednika na wyższym szczeblu i tłumaczy ich elementy. Jest to zatem praktyczne podejście do integracji.

Warstwę Zapobiegającą Uszkodzeniu pomiędzy modelem wyższego i niższego szczebla należy opracowywać, jeżeli tylko jest taka możliwość, tak aby móc tworzyć pojęcia przystające do potrzeb biznesowych danej dziedziny i odizolować się całkowicie od obcych pojęć. Podobnie jednak jak jest w przypadku zatrudnienia tłumacza w roli mediatora, w niektórych przypadkach koszt utworzenia takiej warstwy może być zbyt wysoki.



Usługa Otwartego Hosta

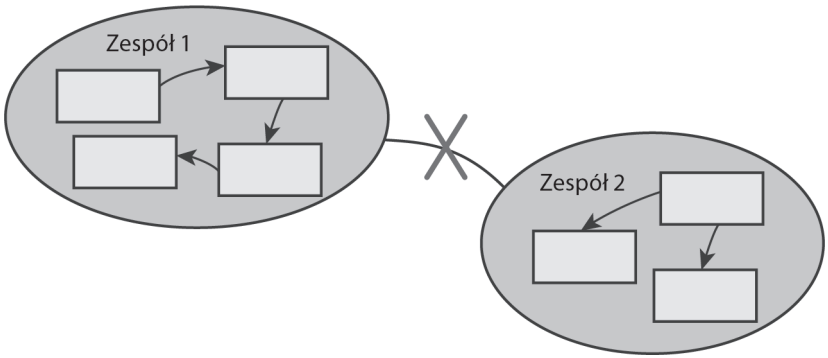
Usługa Otwartego Hosta definiuje protokół lub interfejs, który daje dostęp do Kontekstu Ograniczonego w formie zestawu usług. Protokół jest „otwarty", wobec czego wszyscy, którym zależy na integracji z Kontekstem Ograniczonym, mogą skorzystać z niego względnie bezproblemowo. Usługi oferowane przez API są dobrze udokumentowane i przyjemnie się z nich korzysta. Nawet jeśli należysz do Zespołu 2 z powyższego diagramu i nie masz czasu na stworzenie izolującej Warstwy Zapobiegającej Uszkodzeniu po swojej stronie, to o wiele lepiej być Konformistą w takim modelu niż korzystać z wielu starych systemów, na jakie możesz trafić. Można powiedzieć, że język Usługi Otwartego Hosta jest o wiele prostszy od języków innych rodzajów systemów.



Język Opublikowany

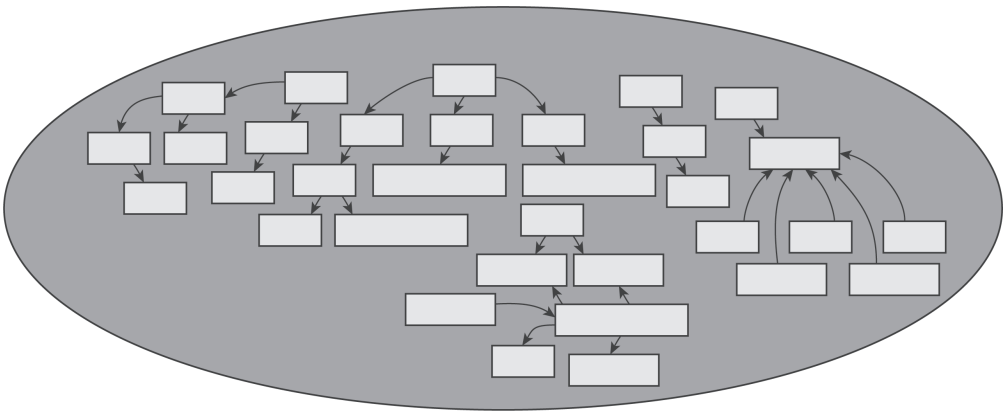
Język Opublikowany, przedstawiony na powyższym rysunku, jest dobrze udokumentowanym językiem do wymiany informacji, umożliwiającym prosty przekaz i tłumaczenie pojęć na dowolnej wielkości zbiór Kontekstów Ograniczonych. Konsumenci mogą tłumaczyć pojęcia na wspólny język i z niego, mając pewność, że integracja terminów jest poprawna. Taki Język Opublikowany

można zdefiniować za pomocą schematu XML, JSON lub w bardziej optymalnych, serializowanych formatach w rodzaju Protobuf lub Avro. Usługa Otwartego Hosta często posługuje się Językiem Opublikowanym, co pozwala zapewnić najlepszą integrację stronom trzecim. Taka kombinacja sprawia, że tłumaczenie dwóch Języków Wszechobecnych staje się bardzo wygodne.



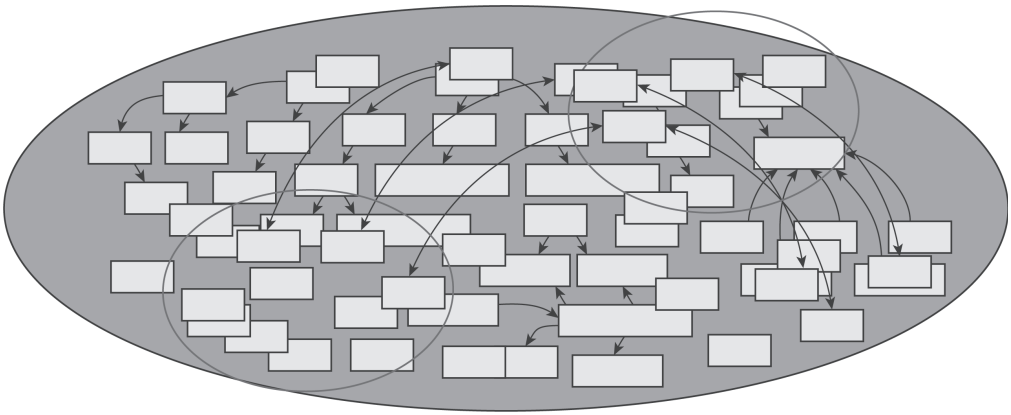
Oddzielne Drogi

Oddzielne Drogi opisują sytuację, w której integracja różnych Kontekstów Ograniczonych i stosowanie wielu Języków Wszechobecnych nie przynosi istotnych korzyści. Jeśli na przykład żaden z Języków Wszechobecnych nie może zapewnić oczekiwanej funkcjonalności, najlepiej stworzyć własne, specjalistyczne rozwiązanie w obrębie swojego Kontekstu Ograniczonego i pominąć integrację na potrzeby tego szczególnego przypadku.

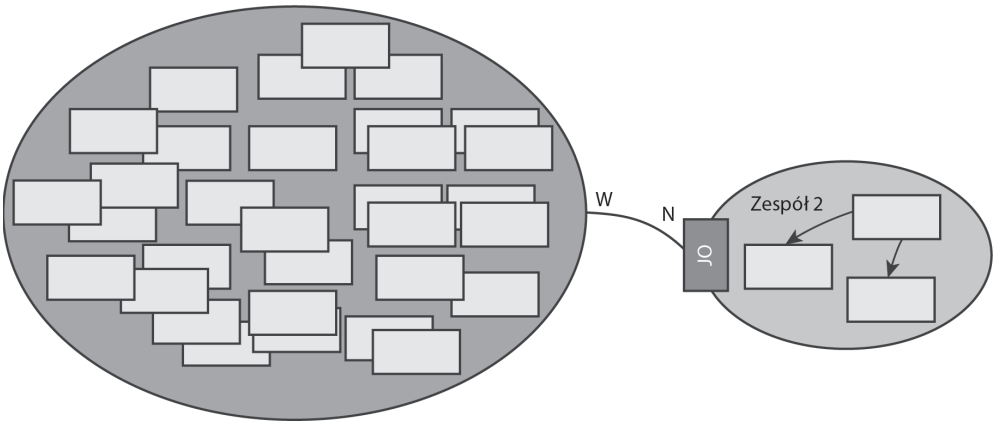


Wielka Kula Błota

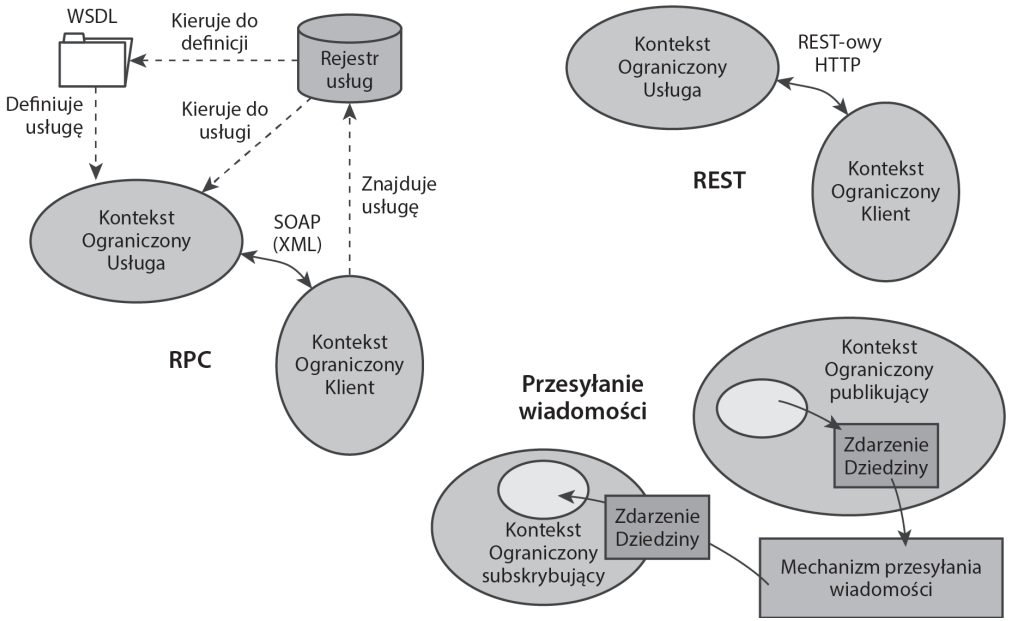
Dowiedziałeś się już całkiem sporo o Wielkiej Kuli Błota w poprzednich rozdziałach, ale chciałbym zwrócić uwagę na problemy, z jakimi trzeba się mierzyć, gdy pracuje się z takim modelem lub dąży się do integracji z nim. Tworzenia Wielkiej Kuli Błota trzeba unikać jak ognia.



Na wszelki wypadek opowiem Ci jeszcze, co się dzieje na dłuższą metę z Wielką Kulą Błota: (1) Coraz więcej Agregatów ulega wzajemnej kontaminacji z powodu nieuzasadnionego tworzenia powiązań i zależności. (2) Konserwacja poszczególnych obszarów Wielkiej Kuli Błota powoduje zniekształcenia w pozostałych, przez co trzeba „gasić pożary" na bieżąco. (3) Jedynie wiedza zachowana przez najstarszych górali i znajomość wszystkich używanych języków pozwala na uratowanie systemu przed całkowitym załamaniem.



Problem w tym, że już w tej chwili istnieje od groma Wielkich Kul Błota i z każdym miesiącem będzie ich przybywać. Nawet jeśli będziesz mógł uniknąć stworzenia Wielkiej Kuli Błota z wykorzystaniem technik DDD, musisz się liczyć z koniecznością przeprowadzenia integracji z takim systemem. W takiej sytuacji powinieneś w miarę możliwości stworzyć Warstwę Zapobiegającą Uszkodzeniom dla każdego starego systemu, tak aby uniknąć zanieczyszczenia własnego modelu niezrozumiałymi śmieciami. Cokolwiek zrobisz — *nie posługuj się takim językiem*!



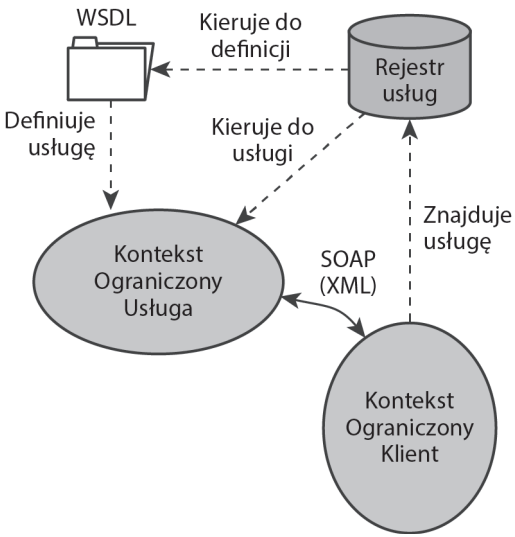
Skuteczne wykorzystanie Mapowania Kontekstu

Być może zastanawiasz się, jaki konkretny interfejs można zastosować w ramach integracji z danym Kontekstem Ograniczonym. Wszystko zależy od tego, co może dostarczyć zespół, do którego taki Kontekst Ograniczony należy. Może to być RPC i SOAP lub REST-owe interfejsy z odpowiednimi zasobami czy też interfejs do przesyłania wiadomości używający kolejek lub modelu Publish-Subscribe. W najmniej korzystnej sytuacji możesz być zmuszony do integracji poprzez bazę danych lub systemów plików, ale miejmy nadzieję, że nie będzie to konieczne. Integracji za pośrednictwem bazy danych należy unikać, ale jeśli już trzeba coś takiego zrobić, to zdecydowanie powinieneś zabezpieczyć model odbiorczy Warstwą Zapobiegającą Uszkodzeniom.

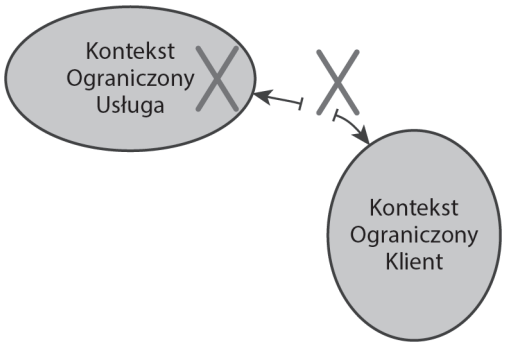
Przyjrzyjmy się trzem najbardziej pewnym rodzajom integracji, kolejno od najmniej do najbardziej niezawodnego. Najpierw przyjrzymy się RPC, następnie REST-owemu protokołowi HTTP, a na koniec przejdziemy do wymiany wiadomości (zobacz pierwszy rysunek na następnej stronie).

RPC i SOAP

Zdalne wywołania procedury (ang. *remote procedure calls* — RPC) mogą działać na różne sposoby. Popularnym sposobem jest używanie RPC za pośrednictwem protokołu SOAP, a polega to na sprawieniu, aby korzystanie z usług z innego systemu wydawało się prostą, lokalną procedurą lub wywołaniem metody. Tym niemniej żądanie SOAP musi zostać przekazane przez sieć, dotrzeć do zdalnego systemu, zostać pomyślnie wykonane, a następnie zwrócić wyniki przez sieć. Wiąże się to z ryzykiem zgubienia informacji wskutek awarii sieci albo przynajmniej wystąpienia

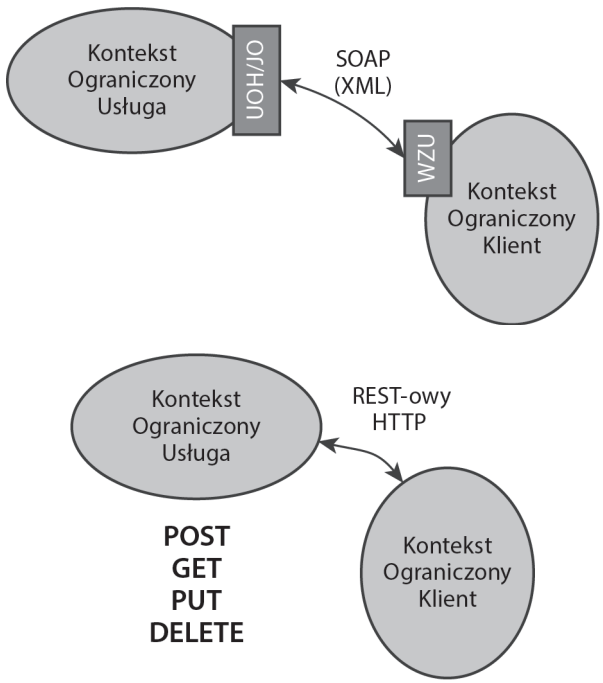


niespodziewanych opóźnień przy pierwszym wdrożeniu integracji. Ponadto wykorzystanie RPC i SOAP wiąże się z silnym sprzężeniem klienckiego Kontekstu Ograniczonego z Kontekstem Ograniczonym, który dostarcza usługę.



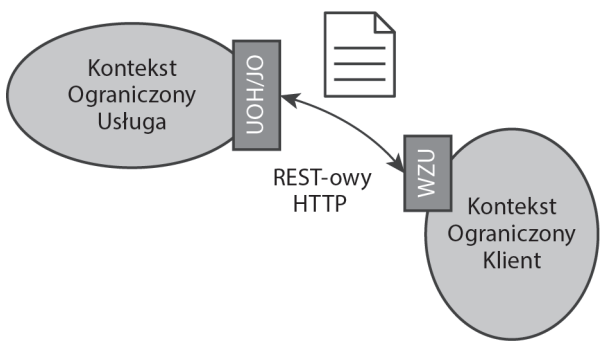
Głównym problemem związanym ze stosowaniem RPC (przy użyciu SOAP lub w inny sposób) jest niestabilność tej metody. W razie wystąpienia jakiegoś problemu z siecią lub systemem hostującym SOAP API teoretycznie proste wywołanie procedury całkowicie zawiedzie, zwracając jedynie błędy. Nie daj się zwieść pozornej prostocie tego rozwiązania (zobacz pierwszy rysunek na następnej stronie).

Kiedy RPC się sprawdza — a jest tak zazwyczaj — to jest bardzo użyteczną metodą integracji. Jeżeli możesz wpłynąć na postać udostępniającego usługę Kontekstu Ograniczonego, to powinno Ci zależeć na stworzeniu dobrze zaprojektowanego API, zapewniającego Usługę Otwartego Hosta z Językiem Opublikowanym. Tak czy inaczej, kliencki Kontekst Ograniczony można uzupełnić o Warstwę Zapobiegającą Uszkodzeniom, aby odizolować model od niepożądanych wpływów z zewnątrz (zobacz drugi rysunek na następnej stronie)..



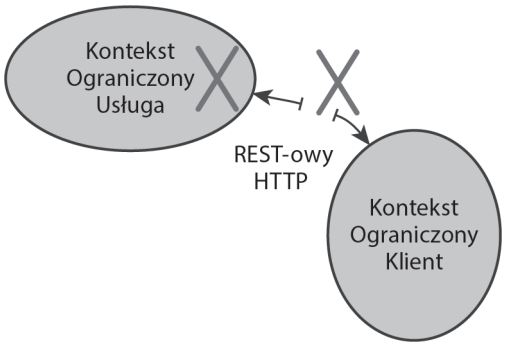
REST-owy HTTP

Integracja przy użyciu REST-owego HTTP koncentruje się na zasobach wymienianych między Kontekstami Ograniczonymi i polega na stosowaniu czterech głównych operacji: `POST`, `GET`, `PUT` i `DELETE`. Korzystanie z REST w integracji jest dla wielu wygodne, gdyż pomaga w tworzeniu dobrych API na potrzeby przetwarzania rozproszonego. Trudno to podważyć, skoro Internet odniósł tak wielki sukces.

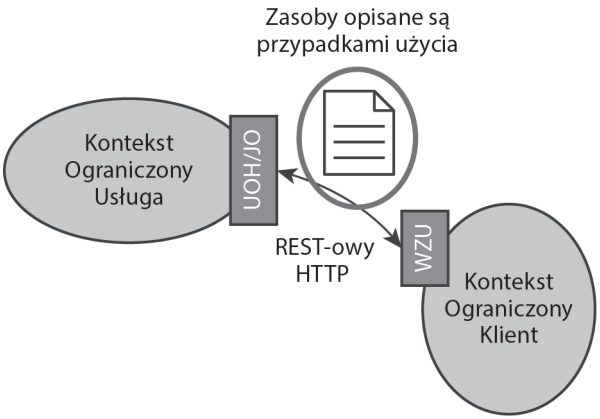


Korzystanie z REST-owego HTTP wiąże się z pewnym określonym sposobem myślenia. Nie opiszę tutaj tego zagadnienia szczegółowo, ale jeśli jesteś zainteresowany, sięgnij po książkę *REST in Practice* [RiP].

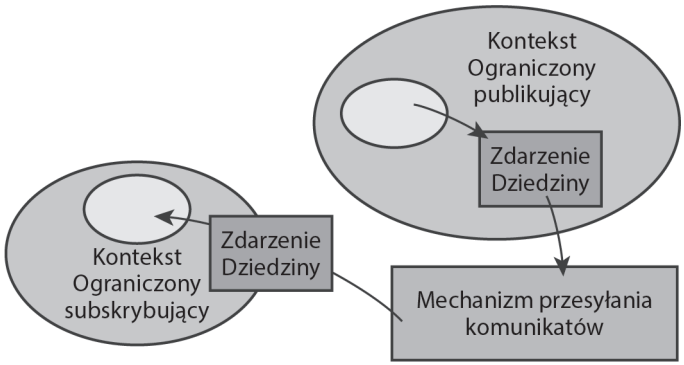
Kontekst Ograniczony usługi, wykorzystujący interfejs REST, powinien zapewniać Usługę Otwartego Hosta i Język Opublikowany. Zasoby powinny być zdefiniowane jako Język Opublikowany, a w połączeniu z URI REST naturalnie przekształcą się w Usługę Otwartego Hosta.



REST-owy HTTP zawodzi przeważnie z tych samych względów co RPC — pojawiają się awarie sieci i usług bądź następuje nieoczekiwane spowolnienie. REST-owy HTTP oparty jest jednak na założeniach Internetu, a w gruncie rzeczy trudno wskazać coś lepszego od sieci WWW pod względem niezawodności, skalowalności i ogólnej użyteczności.



Typowym błędem występującym przy korzystaniu z REST jest projektowanie zasobów tak, aby bezpośrednio odzwierciedlały Agregaty modelu dziedziny. W ten sposób zmuszasz każdego klienta do stania się Konformistą, jako że w przypadku zmiany kształtu modelu zasoby też się zmienią. Zamiast tego zasoby należy projektować syntetycznie, zgodnie z przypadkami użycia u klienta. Przez „syntetyczne" rozumie się to, że klient musi otrzymywać zasoby o kształcie i składzie odpowiadających jego potrzebom, a nie będących odbiciem rzeczywistego modelu dziedziny. Czasami model rzeczywiście odpowiada potrzebom klienta, ale to właśnie one, a nie bieżący kształt, powinny kierować projektowaniem zasobów.



Wymiana wiadomości

Przy stosowaniu asynchronicznej wymiany wiadomości wiele można osiągnąć poprzez sprawienie, aby kliencki Kontekst Ograniczony subskrybował Zdarzenia Dziedziny publikowane przez inny Kontekst Ograniczony. Wykorzystanie obsługi wiadomości jest jedną z najsolidniejszych form integracji, ponieważ umożliwia pozbycie się zależności czasowych występujących w przypadku użycia RPC i REST. Ponieważ od razu możesz przewidzieć wystąpienie opóźnień w wymianie wiadomości, to jesteś przygotowany na tworzenie bardziej stabilnych systemów, jako że nie oczekujesz natychmiastowych wyników.

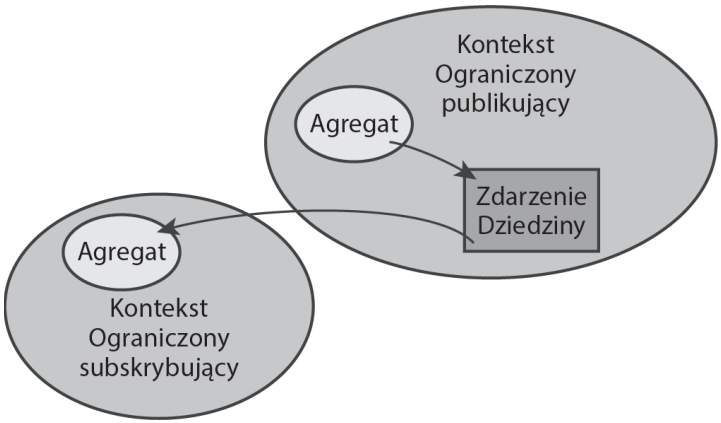
Asynchroniczny REST

Prowadzenie komunikacji asynchronicznej przy użyciu REST, polegające na odpytywaniu rosnącego zbioru zasobów, jest możliwe. Wykorzystując możliwości przetwarzania w tle, klient stale odpytuje usługę, która dostarcza ciągle powiększający się zbiór Zdarzeń Dziedziny. Jest to bezpieczne podejście do prowadzenia operacji asynchronicznych między usługą a klientami przy jednoczesnym dostarczaniu aktualnych zdarzeń ze strony usługi. Jeśli usługa stanie się z jakiegoś powodu niedostępna, klient może ponawiać zapytanie w standardowych interwałach bądź wstrzymać się do momentu, gdy strumień stanie się ponownie dostępny.

To podejście opisane jest szczegółowo w książce *DDD dla architektów oprogramowania* [IDDD].

Unikaj integrowania błędów

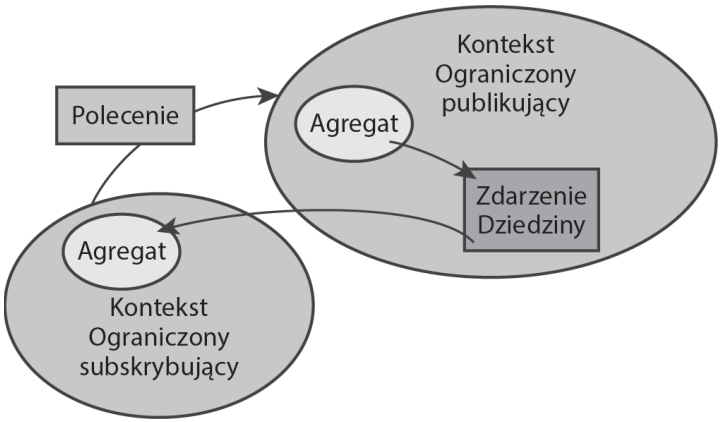
Kiedy kliencki Kontekst Ograniczony (C1) integruje się z Kontekstem Ograniczonym usługi (S1), C1 nie powinien na ogół wysyłać synchronicznego, blokującego żądania do S1 w ramach obsługi żądania. Innymi słowy, dopóki jakiś inny klient (C0) wysyła żądanie blokujące do C1, nie pozwalaj, by C1 wysłał żądanie blokujące do S1. Postąpienie inaczej doprowadziłoby do stworzenia topornej integracji pomiędzy C0, C1 i S1. Zastosowanie asynchronicznej wymiany wiadomości pozwala na uniknięcie tego.



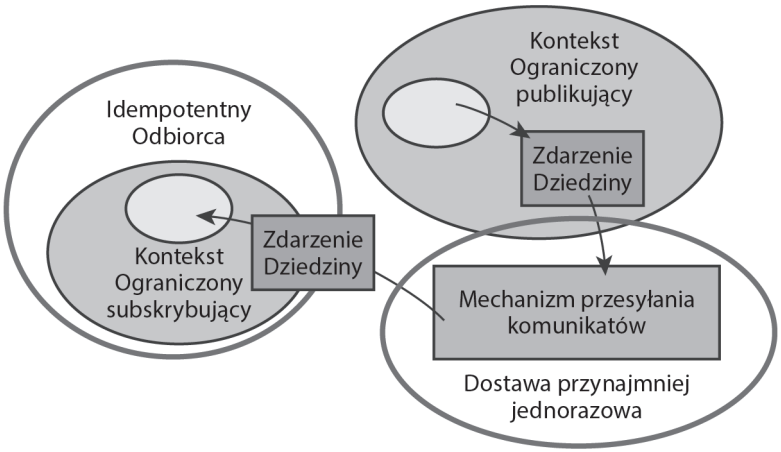
Przeważnie jest tak, że Agregat z jednego Kontekstu Ograniczonego publikuje Zdarzenie Dziedziny, które zostaje przyjęte przez dowolną liczbę zainteresowanych stron. Kiedy subskrybujący Kontekst Ograniczony otrzymuje Zdarzenie Dziedziny, jakieś działania zostają podjęte na podstawie jego typu i wartości. Zazwyczaj powoduje to utworzenie nowego Agregatu lub przekształcenie istniejącego Agregatu w odbierającym Kontekście Ograniczonym.

Czy konsumenci Zdarzeń Dziedziny są Konformistami?

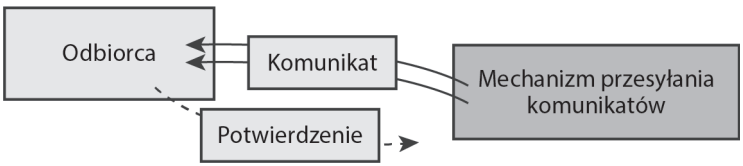
Pewnie zastanawiasz się, czy Kontekst Ograniczony może konsumować Zdarzenia Dziedziny innego Kontekstu Ograniczonego, nie stając się przy tym Konformistą. Jak zaleciłem w *DDD dla architektów oprogramowania* [IDDD], a konkretnie w rozdziale 13. „Integrowanie Kontekstów Ograniczonych", konsumenci nie powinni używać typów zdarzeń (np. klas) producenta zdarzeń, lecz powinni raczej polegać wyłącznie na schemacie zdarzeń, czyli Języku Opublikowanym. Oznacza to na ogół, że jeśli zdarzenia publikowane są jako JSON lub w bardziej ekonomicznym formacie obiektowym, konsument powinien je przetwarzać w celu uzyskania pasujących mu atrybutów danych.



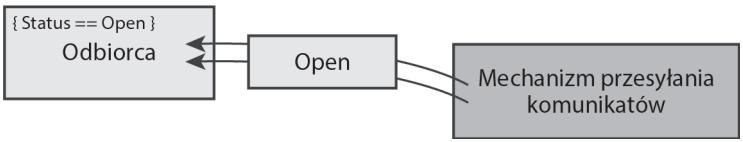
Powyższe zakłada, że subskrybujący Kontekst Ograniczony zawsze korzysta na tym, że w publikującym Kontekście Ograniczonym pojawiają się zdarzenia. Czasami jednak kliencki Kontekst Ograniczony musi sam wysłać Polecenie do Kontekstu Ograniczonego usługi, aby wymusić jakieś działanie. W takich przypadkach kliencki Kontekst Ograniczony również otrzymuje wszelkie rezultaty jako opublikowane Zdarzenia Dziedziny.



We wszystkich przypadkach wykorzystania wymiany wiadomości w integracji jakość rozwiązania w całości zależy w znacznej mierze od jakości wybranego mechanizmu przesyłania wiadomości. Powinien on obsługiwać dostawę „przynajmniej jednorazową" (ang. *At-Least-Once Delivery*) [Reactive], aby można było mieć pewność, że wszystkie wiadomości zostaną ostatecznie odebrane. Oznacza to też, że subskrybujący Kontekst Ograniczony musi być wdrożony jako Idempotentny Odbiorca [Reactive].

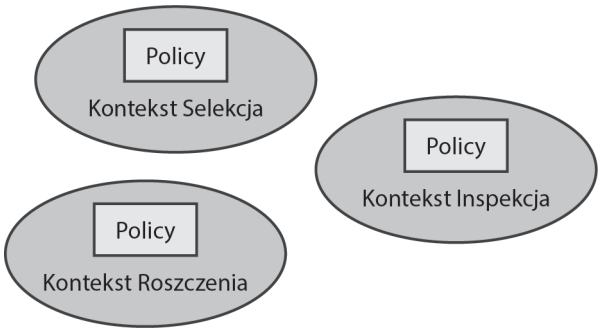


Dostawa „przynajmniej jednorazowa" [Reactive] jest wzorcem przesyłu wiadomości, w ramach którego mechanizm okresowo ponawia wysyłkę danej wiadomości. Dzieje się to w przypadku utraty wiadomości, powolnie reagujących lub nieaktywnych odbiorców, a także tych, którzy nie potwierdzają odbioru. W myśl tego wzorca jedna wiadomość może być wysłana jeden raz, ale dostarczona wielokrotnie. Nie powinno to być jednak problemem, jeśli odbiorca jest zaprojektowany z uwzględnieniem tego zachowania.



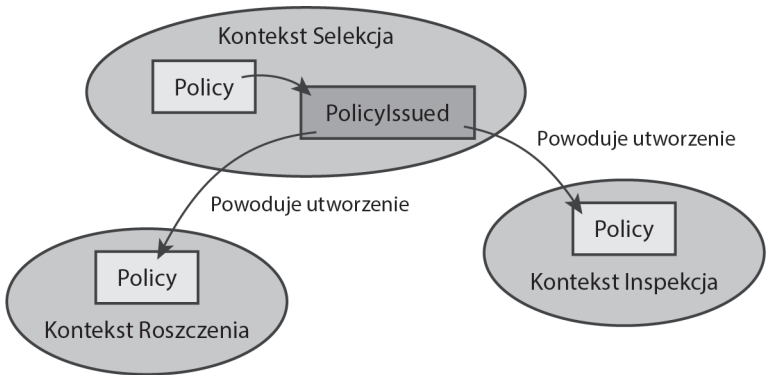
Jeśli wiadomość może być dostarczona więcej niż jednokrotnie, odbiorca powinien być zaprojektowany tak, aby poprawnie zareagować na tę sytuację. Idempotentny Odbiorca [Reactive] to taki, który wykonuje operację w taki sposób, że zawsze zwraca ona jednakowy wynik, nawet jeśli jest wykonywana wielokrotnie. Jeśli zatem ta sama wiadomość zostaje odebrana wielokrotnie, odbiorca obsługuje ją w bezpieczny sposób. Oznacza to, że musi on używać mechanizmu wykrywania zduplikowanych wiadomości i ignorować powtarzającą się wiadomość bądź bezpiecznie zwrócić taki sam wynik jak przy poprzedniej przesyłce.

Ze względu na to, że mechanizmy przesyłania wiadomości zawsze wprowadzają asynchroniczną komunikację opartą na wzorcu Żądanie-Odpowiedź [Reactive], należy się spodziewać pewnych opóźnień. Żądania usług (prawie) nigdy nie powinny blokować wykonania usługi. Projektowanie z myślą o przekazie wiadomości oznacza, że zawsze przewidujesz choćby minimalne opóźnienia, dzięki czemu takie rozwiązanie może być z założenia niezawodne.

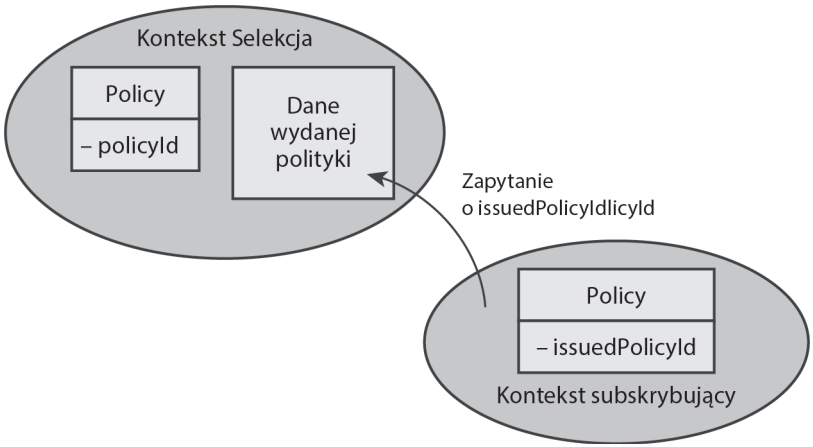


Przykład Mapowania Kontekstu

Wróćmy do przykładu przedstawionego w rozdziale 2. „Projektowanie strategiczne — Konteksty Ograniczone i Język Wszechobecny". Pojawia się teraz pytanie o położenie typu `Policy`. Jak pamiętasz, w trzech różnych Kontekstach Ograniczonych istnieją trzy rodzaje `Policy`. Gdzie zatem umiejscowić „politykę rejestracji" w firmie ubezpieczeniowej? Możliwe, że powinna być przypisana do działu selekcji, ponieważ to on się nią zajmuje. Na potrzeby przykładu załóżmy, że tak jest. Jak zatem inne Konteksty Ograniczone dowiadują się o jej istnieniu?



Kiedy komponent typu `Policy` zostaje wydany w Kontekście Selekcja, może on opublikować Zdarzenie Dziedziny o nazwie `PolicyIssued`. Dowolny inny, subskrybujący Kontekst Ograniczony może zareagować na Zdarzenie Dziedziny, np. tworząc odpowiadający mu komponent `Policy` w swoim obrębie.



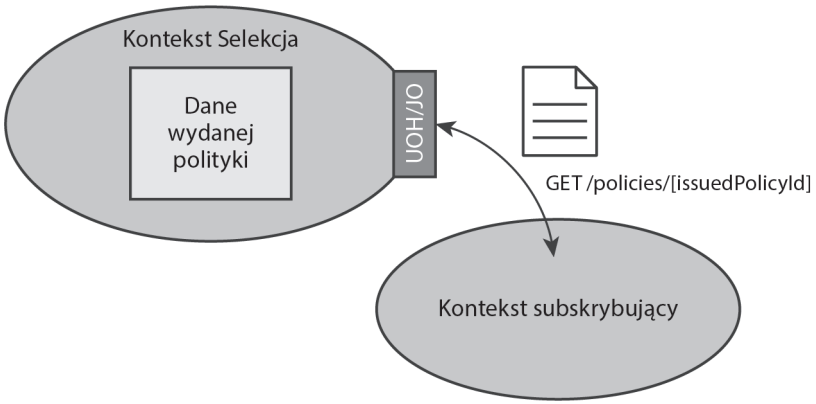
Zdarzenie Dziedziny `PolicyIssued` zawiera zapis o tożsamości komponentu `Policy`. W tym przypadku jest to `policyId`. Wszelkie komponenty utworzone w subskrybującym Kontekście Ograniczonym zachowałyby tę informację, aby można było prześledzić, że pochodzą z Kontekstu Selekcja. W tym przykładzie tożsamość zapisana jest jako `issuedPolicyId`. Jeśli potrzeba więcej danych niż te, które dostarczyło Zdarzenie Dziedziny `PolicyIssued`, subskrybujący Kontekst Ograniczony zawsze może ponownie wysłać zapytanie do Kontekstu Selekcja, aby uzyskać ich więcej. W tym przypadku subskrybujący Kontekst Ograniczony używa `issuedPolicyId` do przeszukania Kontekstu Selekcja.

Wzbogacanie zdarzeń a ponawianie zapytań

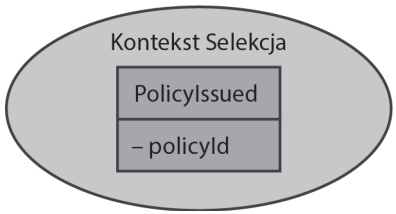
Czasami korzystnym rozwiązaniem jest wzbogacanie Zdarzeń Dziedziny o dane pozwalające na zaspokojenie potrzeb wszystkich konsumentów. Bywa też jednak, że lepiej jest tworzyć minimalistyczne Zdarzenia Dziedziny i pozwalać na wydawanie zapytań zwrotnych, gdy konsumenci potrzebują dodatkowych danych. Wzbogacenie zapewnia zależnym konsumentom większą autonomię. Jeśli to autonomia jest kluczowym wymaganiem, rozważ wykorzystanie tego rozwiązania.

Z drugiej strony, trudno przewidzieć wszystkie rodzaje danych, jakich konsumenci będą potrzebować od Zdarzeń Dziedziny, a dostarczenie ich wszystkich jest niepraktyczne. Na przykład nadmierne wzbogacenie Zdarzeń Dziedziny może być zagrożeniem dla bezpieczeństwa. Jeśli tak jest, to przygotowanie minimalistycznych Zdarzeń Dziedziny i skonstruowanie rozbudowanego, lecz bezpiecznego modelu zapytań dla konsumentów może okazać się lepszym rozwiązaniem.

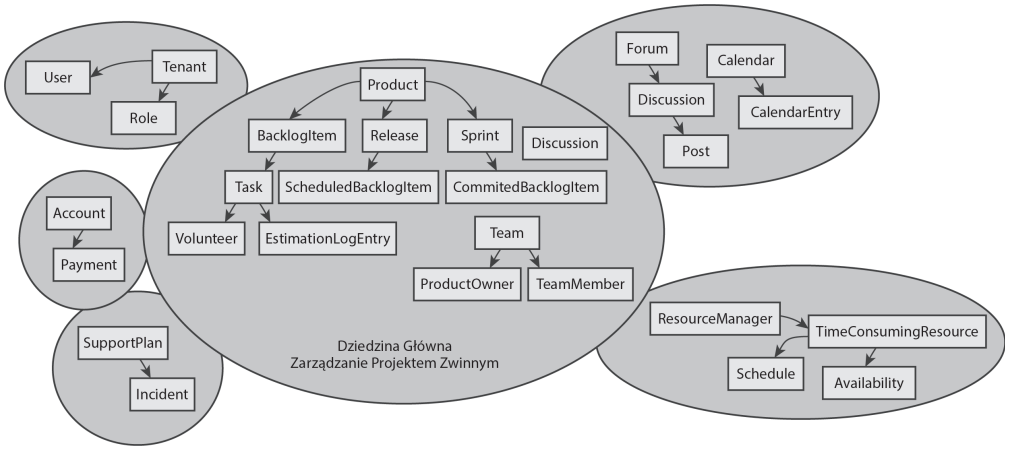
Czasami okoliczności wymagają opracowania wyważonego rozwiązania, wykorzystującego elementy obydwu tych podejść.



Jak miałoby wyglądać wysłanie zapytania zwrotnego do Kontekstu Selekcja? W tym przypadku dla Kontekstu Selekcja można stworzyć Usługę Otwartego Hosta i Język Opublikowany z zasobów REST. Wystarczy prosta operacja HTTP `GET` z wykorzystaniem `issuedPolicyId`, aby uzyskać `IssuedPolicyData`.



Zapewne zastanawiasz się, jakie konkretnie dane przekazywane są w Zdarzeniu Dziedziny `PolicyIssued`. Szczegółami projektowania Zdarzeń Dziedziny zajmę się w rozdziale 6. „Projektowanie taktyczne — Zdarzenia Dziedziny”.



Być może zastanawiasz się, dlaczego gdzieś po drodze zgubiliśmy Kontekst Zarządzanie Projektem Zwinnym i zamiast tego zajęliśmy się omówieniem dziedzin biznesowych z branży ubezpieczeń. Było to celowe — dzięki temu mogliśmy posłużyć się bardziej zróżnicowanymi przykładami przy omawianiu DDD. Nie martw się jednak — do Kontekstu Zarządzanie Projektem Zwinnym powrócimy w następnym rozdziale.

Podsumowanie

W tym rozdziale dowiedziałeś się o:

- różnych rodzajach relacji w Mapowaniu Kontekstu, takich jak Partnerstwo, Klient-Dostawca i Warstwa Zapobiegająca Uszkodzeniu;
- integracji Mapowania Kontekstu za pomocą RPC, REST-owego HTTP i przesyłu komunikatów;
- wykorzystaniu Zdarzeń Dziedziny w kontekście wymiany wiadomości;
- podstawach, które pomogą Ci zdobyć doświadczenie w zakresie Mapowania Kontekstu.

Obszerne omówienie Map Kontekstu znajdziesz w rozdziale 3. książki *DDD dla architektów oprogramowania* [IDDD].

Skorowidz

O

P

R

S

T

U

W

Z

Narzędzia DDD — ciesz się udanym wdrożeniem!

Modelowanie oprogramowania jest kojarzone z pojęciem programowania dziedzinowego, w skrócie zwanego DDD. Jest to dość nowatorskie podejście do tworzenia architektury oprogramowania, a jego największą zaletą jest wysoka pewność uzyskiwania bardzo dobrych rezultatów. Programiści często błyskawicznie dostrzegają zalety DDD i korzyści płynące ze stosowania odpowiednich narzędzi, stąd technika ta cieszy się coraz większym uznaniem. Dopiero jednak pełne zrozumienie zasad stosowania wzorców projektowych DDD przez wszystkich zaangażowanych w projekt pozwala na osiągnięcie imponujących wyników przy projektowaniu skomplikowanych systemów oprogramowania.

Ten zwięzły i czytelnie napisany podręcznik jest przeznaczony dla programistów, ekspertów dziedzinowych, menedżerów, analityków biznesowych, architektów informacji i testerów. Koncentruje się na praktycznej wiedzy niezbędnej do uzyskania pożądanych rezultatów. Wyjaśniono, jak segregować modele dziedzin za pomocą wzorca kontekstu ograniczonego, jak rozwinąć język wszechobecny, a także w jaki sposób zintegrować wiele kontekstów ograniczonych poprzez wykorzystanie relacji zespołowych i mechanizmów technicznych. Co najważniejsze, dowiesz się, jak podejście DDD działa w rzeczywistości i co zrobić, aby jak najszybciej cieszyć się jego zaletami.

W książce przedstawiono następujące zagadnienia:

- wprowadzenie do DDD i zalety tego podejścia do projektowania
- projektowanie strategiczne DDD
- integrowanie istniejących systemów w ramach tworzenia nowych aplikacji
- modelowanie taktyczne — agregaty i zdarzenia dziedziny
- narzędzia do zarządzania projektami i przyśpieszania prac

Vaughn Vernon jest uznanym liderem nowatorskiego podejścia do implementacji oprogramowania. Zasady programowania dziedzinowego stosuje w praktyce, tworząc modele oprogramowania dla takich branż jak zarządzanie przestrzenią powietrzną, ochrona środowiska, ubezpieczenia, ochrona zdrowia czy telekomunikacja. Jest uznanym autorytetem w dziedzinie DDD — prowadzi warsztaty Implementing DDD Workshop na całym świecie z udziałem setek programistów. Jest autorem kilku bestsellerów. Często występuje na prestiżowych konferencjach branżowych.

ISBN 978-83-283-4279-8
9 788328 342798
Cena: 34,90 zł
INFORMATYKA W NAJLEPSZYM WYDANIU
Addison-Wesley
ALWAYS LEARNING
PEARSON