INFRASTRUKTURA INFORMACJI PRZESTRZENNEJ W UML

Wydawca
Geodeta Sp. z o.o.
ul. Narbutta 40/20
02-541 Warszawa
www.geoforum.pl
redakcja@geoforum.pl

Opracowanie redakcyjne,
skład i łamanie
Geodeta Sp. z o.o.
www.geoforum.pl

Projekt graficzny i okładka
Andrzej Rosołek

Korekta językowa
Hanna Szamalin

Druk
Drukarnia Taurus

ISBN 978-83-935447-1-4

INFRASTRUKTURA INFORMACJI PRZESTRZENNEJ W UML

Zenon Parzyński, Agnieszka Chojka

GEODETA
WYDAWNICTWO
Warszawa 2013

Chciałbym w tym miejscu serdecznie podziękować wszystkim, którzy byli ze mną w trakcie pisania tej książki, a także tym, którzy zaryzykowali, podjęli się jej przeczytania i mieli mniej lub więcej uwag, co spowodowało, że efekt końcowy jest zdecydowanie lepszy.
Szczególnie chcę wyrazić wdzięczność za liczne i cenne uwagi p. Grażynie Skołbani.
Korzystając z okazji, chciałbym też publicznie podziękować „moim" trzem profesorom, których miałem szczęście spotkać na swojej drodze:

- prof. dr. hab. inż. Zdzisławowi Adamczewskiemu – który m.in. „troszkę" mnie przećwiczył w ramach prac statutowych, co wyszło mi tylko na zdrowie;
- prof. dr. hab. inż. Witoldowi Prószyńskiemu – który zgodził się zostać promotorem mojej pracy doktorskiej i który w dalszym ciągu daje się zapraszać na pogawędki z kawą w tle;
- prof. dr. hab. inż. Wojciechowi Pachelskiemu – który był, jest i, mam nadzieję, że nadal będzie moim nauczycielem modelowania pojęciowego, UML, standaryzacji, norm ISO serii 19100 itp.

Serdecznie Wam dziękuję – jesteście ważnymi osobami w moim życiu

Zenon Parzyński

Spis treści

1. Wstęp

Skąd pomysł na taką książkę? Otóż od paru lat geoinformatyka robi oszałamiającą karierę. Ta kariera jest podbudowana możliwościami technicznymi, jakie daje informatyka. Można operować coraz większą ilością danych i nie trzeba czekać godzinami na efekty. Można realnie zacząć myśleć o rozwiązywaniu problemów obejmujących swoim zasięgiem duże rejony kuli ziemskiej. Na temat geoinformatyki publikuje się coraz więcej artykułów i podręczników. Jednym z przykładów jest książka pod redakcją czterech Chińczyków (Yang Ch. i in., 2011). Twierdzą oni – i według mnie mają rację – że ostatnio wiele spraw o zasięgu ogólnoświatowym lub regionalnym zyskało na znaczeniu, stając się wyzwaniem dla obywateli świata. By stawić im czoła, niezbędna jest współpraca, która przekracza granice państw czy różnych organizacji. Te sprawy mają bardzo często wymierny aspekt fizyczny wpływający na ludzi i środowisko w skali regionalnej (jak tsunami czy huragany) lub globalnej (jak ocieplenie klimatu). Takiej współpracy wymagają też występujące kryzysy ekonomiczne, epidemie itp.

W literaturze można znaleźć wiele pozycji (m.in. Gotlib D. i in., 2007; Bielecka E., 2006) pokazujących różnorodne sfery, w których informacja przestrzenna (IP) jest użyteczna, wykorzystywana, a obecnie wręcz niezbędna. Oprócz „zwykłych" zastosowań (jak bazy topograficzne, rejestr granic, system identyfikacji działek rolnych, administracja, systemy szybkiego reagowania, leśnictwo czy systemy przeciwpowodziowe) pojawiają się także te na pierwszy rzut oka niezbyt oczywiste (jak analiza ryzyka ubezpieczeń, akcje charytatywne czy ochrona zdrowia).

Drogą do uzyskania informacji będących podstawą racjonalnych decyzji jest geoinformatyka. Umożliwia ona przeprowadzenie różnego rodzaju analiz wielowymiarowych. Wiele wymiarów uzyskamy, nakładając na informacje czasowo-przestrzenne informacje dodatkowe (opisowe). Analizując takie dane, można spróbować uzyskać nowe informacje, wyciągać wnioski, np. jak zaradzić jakiemuś niebezpieczeństwu. Takie analizy robiono jeszcze przed erą informatyki. O tym, że zagrożenie lawinowe w ewidentny sposób zależy od grubości pokrywy śnieżnej, wiedziano od dawna. Ale to dopiero geoinformatyka – wykorzystując i łącząc w jeden system osiągnięcia informatyki, geodezji, inżynierii środowiska itp. – pozwala objąć analizą duże obszary i wiele zjawisk jednocześnie.

Wydawałoby się, że nic prostszego, jak zebrać dane, dać mądrym ludziom do analizy i wdrożyć w życie ich wnioski. Jednak, pomijając pewne ludzkie słabości, sama idea łączenia różnych informacji jest bardzo kłopotliwa w praktycznej realizacji. Do połączenia np. danych o położeniu (czyli gdzie się znajduje ten kawałek powierzchni Ziemi, którego dotyczy dana informacja) potrzeba wielu skoordynowanych działań.

Różnego rodzaju informacje są przechowywane w bazach jako dane. Każda baza danych musi mieć zdefiniowaną strukturę. W tej definicji określa się, jakie informacje i o jakich obiektach będą w bazie gromadzone, a także jakie połączenia występują między poszczególnymi obiektami. Do opracowania modelu struktury bazy danych używa się różnych metod – jedną z nich jest modelowanie pojęciowe. W wyniku modelowania powstaje oczywiście model, który to model trzeba zapisać w jakimś języku, np. w UML. I tu dochodzimy do sedna sprawy – czyli po co to?

Niniejsza książka ma za zadanie pomóc w nauce czytania modeli pojęciowych zapisanych w języku UML (*Unified Modeling Language* – Zunifikowany Język Modelowania). Chodzi oczywiście o czytanie ze zrozumieniem! To wyraźnie zaznaczam, bo plany mam niezwykle ambitne, a podobno z tym zrozumieniem różnych rzeczy dużo osób ma coraz więcej problemów, nie wyłączając mnie samego. Język UML jest standardem „praktycznym", ponieważ nie został ustanowiony formalnie przez jakąś instytucję, np. Unię Europejską, ale jest powszechnie wykorzystywany. W coraz większej liczbie aktów prawnych w Polsce zaczynają się pojawiać modele zapisane w tym języku.

W jednej z książek o UML przeczytałem, że większości potencjalnych użytkowników z reguły brakuje czasu, wiedzy lub motywacji do poznawania w pełnym zakresie narzędzia o takiej skali możliwości (Graessle P. i in., 2006).

Możesz, Szacowny Czytelniku, założyć, że to stwierdzenie jest ze wszech miar słuszne, bo mnie też zabrakło motywacji do poznania całego UML. Ten, o którym będę pisał, jest UML-em w znacznym stopniu ograniczonym. Ograniczeniu podlega liczba omawianych diagramów, z których składa się UML. Będę omawiał jedynie diagram klas, a z niego tylko elementy najczęściej wykorzystywane w geoinformatyce.

Zasadne też jest pytanie: A dlaczego UML? Otóż dlatego, że:

- jest językiem utworzonym do zapisywania modeli (głównie systemów, ale nie tylko),
- jest coraz częściej wykorzystywany,
- został użyty przy tworzeniu m.in. norm ISO serii 19100 dotyczących informacji geograficznej,
- zaczął się „pokazywać" w różnych aktach prawnych, jak np. rozporządzenia opracowane w GUGiK (Główny Urząd Geodezji i Kartografii) czy w GDOŚ (Generalna Dyrekcja Ochrony Środowiska), a w przyszłości różnych modeli UML będzie przybywać,
- jest podstawą zapisu modelu w języku GML jako plików XSD (więcej o plikach XSD w rozdziale 8).

Jak Państwo widzą, powodów jest sporo.

W naszej książce wiele terminów angielskich nie będzie tłumaczonych. Jest to postępowanie zamierzone. Głównie chodzi o to, by uniknąć trudności. Po co mamy się głowić i wymyślać, jak przetłumaczyć np. nazwy typów *Truth, Boolean, Logical* z rysunku A.2.2. znajdującego się w dodatku A? Wszystkie one opisują typy logiczne i my mamy na to jedną nazwę. Do podobnej konkluzji, że wielu terminów angielskich nie będzie się tłumaczyć na język polski, doszli specjaliści w Polskim Komitecie Normalizacyjnym (choć zapewne ich pobudki były inne). Często będę podawał znaczenie pojęć, np. *Integer* oznacza liczbę całkowitą, ale w dokumentach jako nazwa będzie używany termin angielski.

Jak podejść do tej książki? Najlepiej na spokojnie, w fotelu, z filiżanką kawy lub herbaty. W rozdziale 2 próbowałem skonkretyzować parę pojęć, których znajomość pomoże w czytaniu książki. O wielu z nich można oczywiście przeczytać w licznych pozycjach, np. w książce, którą ostatnio wydał prof. Wojciech Pachelski wraz ze współautorkami (Pachelski W. i in., 2012). Dla osób nieznających UML i modelowania pojęciowego niezbędne będą rozdziały 3 i 4. Można na początku pominąć rozdział 5 o normach i przejść do dyrektywy INSPIRE (rozdział 6). Do norm (rozdział 5) można wrócić, gdy podczas czytania pojawią się na schematach nazwy klas zdefiniowanych w jakiejś normie. W rozdziale 7 podaję kilka konkretnych przykładów i koncepcji wziętych prosto z życia. Na końcu jest rozdział o GML (*Geography Markup Language*), który ma pełnić funkcję standardu wymiany danych. Rozdział ten zgodziła się napisać dr inż. Agnieszka Chojka (ja o GML wiem tyle, że istnieje i w zasadzie jest przydatny). Można też podejść do książki standardowo (ostatecznie pewna jej część dotyczy standardów) i przeczytać wszystko od początku do końca.

Muszę uprzedzić tych z Państwa, którzy będą chcieli porównać modele z książki z modelami np. z norm, że mogą pomiędzy nimi wystąpić drobne różnice. Opierałem się głównie na zbiorze z modelami UML „2010-04-26_r937.EAP", który skopiowałem ze stron INSPIRE. Czasem też niektóre modele zostały przeze mnie trochę „odchudzone" (zawierają w książce mniej klas niż w oryginale). Uważam, że te różnice nie wpływają na właściwy odbiór i zrozumienie modeli oraz przeznaczenia klas w nich umieszczonych.

Pozwolę sobie jeszcze wyrazić nadzieję, że nie będziecie Państwo żałowali wydanych pieniędzy i czasu poświęconego na czytanie.

2. Po trochu o wszystkim

Chcąc pisać o modelowaniu informacji geograficznej i zapisywaniu utworzonych modeli w UML-u, należy zacząć od zdefiniowania (a przynajmniej podjęcia próby zdefiniowania) podstawowych pojęć, takich jak: informacja, informacja przestrzenna, informacja geograficzna, dane, dane geograficzne, bazy danych czy systemy informacyjne i informatyczne.

2.1. Czym jest informacja, a czym są dane?

Z informacją jest kłopot. Różni autorzy różnie to pojęcie definiują. Św. Augustyn napisał kiedyś coś takiego: „Wiem, czym jest czas, dopóki ktoś mnie o to nie zapyta". Podobnie jest z informacją. Trochę w tę stronę zmierza definicja, a może raczej próba jakiegoś opisania informacji przez Norberta Wienera, amerykańskiego matematyka, twórcę cybernetyki: „**Informacja jest informacją**, a nie sprawą energii". W książce „Information Modeling" (Schenck D i in., 1994) zaproponowano definicję, że informacja jest to wiedza o ideach, faktach i/lub procesach. Natychmiast oczywiście nasuwają się pytania: czym jest wiedza, a czym jest idea? W Wikipedii przeczytamy natomiast: „Informacja (łac. *informatio* – przedstawienie, wizerunek*; informare* – kształtować, przedstawiać) – termin interdyscyplinarny, definiowany różnie w różnych dziedzinach nauki; najogólniej – właściwość pewnych obiektów, relacja między elementami zbiorów pewnych obiektów, której istotą jest zmniejszenie niepewności (nieokreśloności)" (Wiki, 2012). Z kolei prof. Jerzy Gaździcki pisze: „Informacja *information*, wiedza uzyskiwana w drodze interpretacji danych, która w ustalonym kontekście ma określone znaczenie i dotyczy obiektów, takich jak fakty, zdarzenia, przedmioty, zjawiska, procesy i idee" (Gaździcki J., 2003).

Zgodnie z normą PN-EN ISO 19101 (ISO 19101, 2002) informacja to „wiedza dotycząca obiektów, takich jak: fakty, zdarzenia, rzeczy, procesy, pojęcia, koncepcje, która w określonym kontekście ma konkretne znaczenie". Polski Komitet Normalizacyjny przyjął, że definicja informacji, ale w przetwarzaniu informacji, jest dokładnie taka sama jak w ISO 19101 (PKN, 1999). Ja jestem zdecydowanym zwolennikiem uznania pojęcia „informacja" za pojęcie pierwotne – niedefiniowalne.

Niezależnie od tego, czy i jak będziemy informację definiować, **informacja geograficzna** jest „połączona" z Ziemią. To połączenie ma oznaczać, że wiemy, jakiego fragmentu Ziemi (lub całego globu) informacja dotyczy. **Informacja przestrzenna** opisuje jakiś fragment przestrzeni (niekoniecznie związanej z Ziemią).

Istnieje jeszcze jeden aspekt informacji, który dotyczy jej **ważności** (**istotności**). Informacją jest, że ktoś ma zielony sweterek. Ale nie jest to informacja istotna, bo nie wiemy, kto go ma, nie wiemy, czy go nosi (może nie nosić), w jakich sytuacjach nosi? Gdybyśmy wiedzieli, kto ma ten sweterek, to już moglibyśmy zacząć się zastanawiać nad np. informacją przestrzenną dotyczącą sweterków, bo jeśli wiemy kto, to najczęściej też wiemy, gdzie ten ktoś mieszka. Nadal jednak pozostaje kwestia: na ile taka informacja jest istotna? Dla tych, którzy żyją ze sweterków (produkcji, sprzedaży itp.), jest to informacja w jakimś sensie ważna, ale znacznie bardziej istotna jest informacja o liczbie sprzedanych zielonych sweterków, a nie, że ktoś taki sweterek ma.

Jak z ogromu informacji, które nas codziennie zalewają, wydobyć te ważne (ja nic nie mówię o ich zapamiętaniu!) i jakoś rozsądnie spożytkować? Jakością informacji zajmował się polski naukowiec prof. Marian Mazur, ale jego rozważania (Mazur M., 1970) nie odbiły się szerokim echem ani w Polsce, ani w świecie, a przecież kwestia ważności informacji jest sprawą ważną!

Czym są **dane**? We wspomnianej już książce „Information Modeling" (Schenck D. i in., 1994) znajdziemy definicję: „Danymi są ciągi symboli reprezentujące informację dla celów przetwarzania i przekazywania (komunikowania) i wykorzystujące ukryte bądź jawne reguły interpretacyjne". Prof. Gaździcki proponuje z kolei tak: „Dane *data*, reprezentacja informacji, właściwa do komunikowania się, interpretacji lub przetwarzania. Dane występują w postaci znaków (...) zrozumiałych dla człowieka lub nadających się do przetwarzania komputero-

wego oraz transmisji" (Gaździcki J., 2003). Według normy PN-EN ISO 19101 (ISO 19101, 2002) dane to „interpretowalna i sformalizowana reprezentacja informacji, stosowna dla jej komunikowania, interpretowania lub/i przetwarzania". Jak widać, w przypadku danych – w przeciwieństwie do informacji – takiego zamieszania definicyjnego nie ma. **Dane geograficzne** reprezentują informację geograficzną, a **dane przestrzenne** informację przestrzenną.

Jest jeszcze jedna kwestia, którą tutaj szerzej zajmować się nie będziemy. Otóż – **co było pierwsze**? Informacja czy dane? Czy wydobywamy (odczytujemy) informację z danych (muszą więc być dane, by móc otrzymać z nich informację), czy też dane są zapisem informacji (więc musi najpierw istnieć informacja, by mogła zostać zapisana w postaci danych)? Nasuwa się tu skojarzenie z nierozwiązaną (jak mi się wydaje) sprawą, co było pierwsze: kura czy jajko?

2.2. Model, system, baza danych

W rozdziale o języku UML wielokrotnie używam słowa **model** i zacząłem się zastanawiać, czy aby wiem, co ono oznacza. Modelem jest „system założeń, pojęć i zależności między nimi pozwalający opisać (zamodelować) w przybliżony sposób jakiś aspekt rzeczywistości" (Wiki, 2012). Powinien zostać zapisany w jakimś formalnym języku, najczęściej w języku matematyki. My będziemy zapisywać model w UML-u. Tak rozumiem model i w tym znaczeniu będzie to słowo używane w książce. Na marginesie można wspomnieć, że trudno jest zrobić dobry model. Powinien on zadowolić odbiorcę (który nam płaci), powinien też być zgodny z kanonami modelowania i języka, w którym został zapisany, i musi spełnić zaplanowaną dla niego funkcję (Graessle P. i in., 2006), wymagań jest więc sporo. Nie można też zbudować dobrego i użytecznego modelu bez znajomości zagadnienia, które będzie modelowane! To jest kolejne utrudnienie w modelowaniu.

Model zapisany w UML-u jest **abstrakcyjnym** przedstawieniem pewnej sytuacji. W naszym przypadku będzie to struktura bazy danych. Przymiotnik „abstrakcyjny" nie oznacza tu: niezwiązany z życiem, z rzeczywistością (jak abstrakcja jest definiowana w „Słowniku wyrazów obcych i zwrotów obcojęzycznych" Kopalińskiego), ale raczej opisuje pewne uogólnienie, pomijanie danych (informacji) nieistotnych dla przedstawianego zagadnienia.

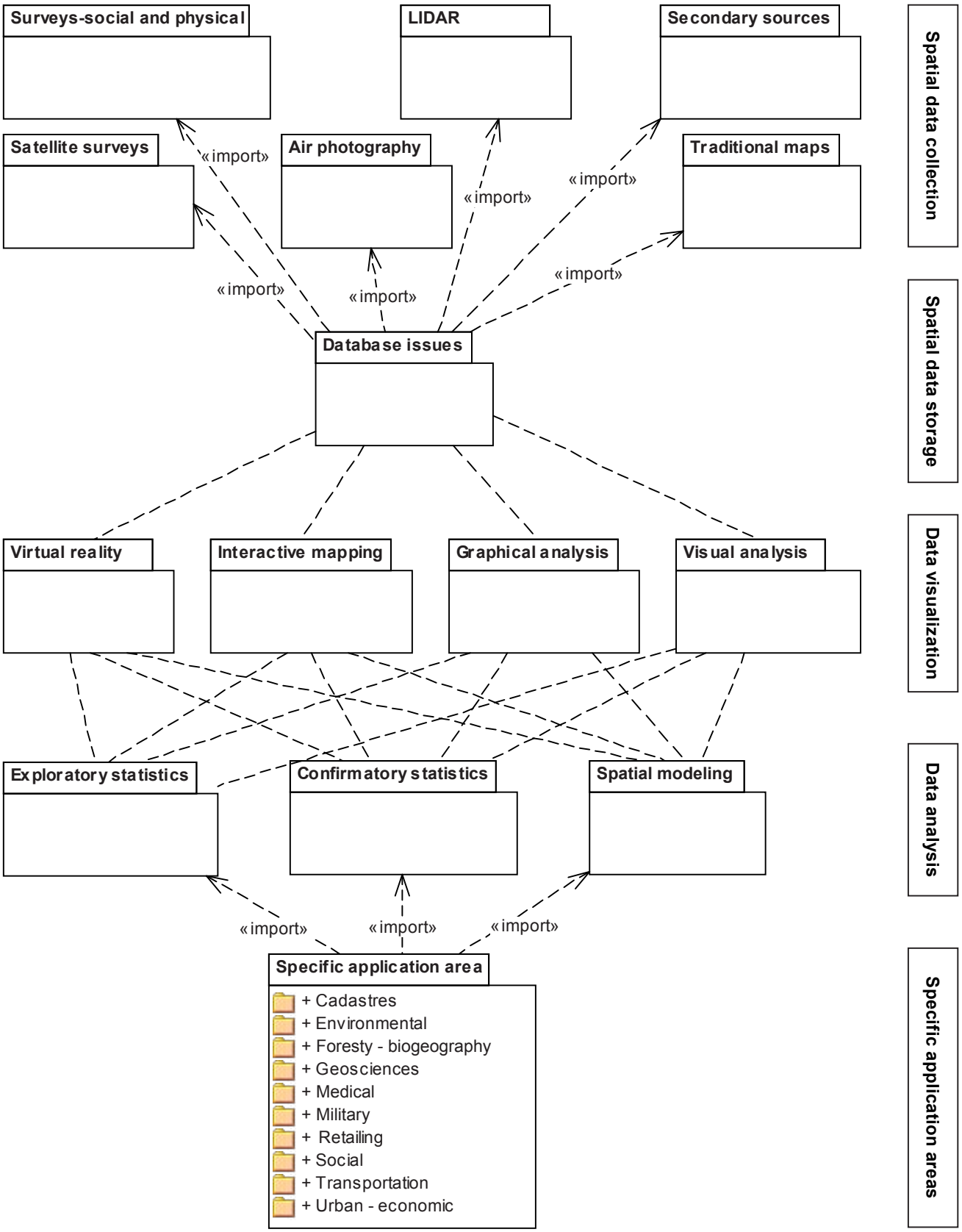
Struktura bazy danych zapisana w postaci modelu w UML-u jest częścią pewnego systemu. **System** zaś jest połączeniem części według jakiejś zasady w jedną całość, która ma spełniać jakąś funkcję, coś sobą reprezentować, coś wykonywać itp. Wyróżnia się bardzo dużo systemów: od filozoficznych (co znaczniejszy filozof – to od razu nowy system), przez różne systemy informacyjne, informatyczne, systemy baz danych, po GIS, SIP, SIT itd. W Wikipedii znalazłem jedną z bardziej ogólnych definicji **systemu informacyjnego**: „System informacyjny to posiadająca wiele poziomów struktura pozwalająca użytkownikowi na przetwarzanie, za pomocą procedur i modeli, informacji wejściowych w wyjściowe" (Wiki, 2012). Z takiego punktu widzenia np. człowiek jest jak najbardziej systemem informacyjnym, pod warunkiem że nie będziemy się zastanawiali nad sensownością informacji wyjściowych. Z kolei **systemem informatycznym** będzie „wydzielona, skomputeryzowana część systemu informacyjnego" (to też z Wikipedii).

Prof. Gaździcki definiuje **system informacji przestrzennej** (*spatial information system*) jako „system pozyskiwania, przetwarzania i udostępniania danych, w których zawarte są informacje przestrzenne oraz towarzyszące im informacje opisowe o obiektach wyróżnionych w części przestrzeni objętej działaniem systemu" (Gaździcki J., 1990). Nic dodać, nic ująć.

2.3. Geomatyka – geoinformatyka – geodezja

Jakie są związki pomiędzy dyscyplinami naukowymi, których elementy tworzą geomatykę? Dobrze byłoby wyrobić sobie zdanie w tej kwestii. W 2008 r. PWN wydało podręcznik prof. Stefana Przewłockiego zatytułowany „Geomatyka" (Przewłocki S., 2008). W tytułach rozdziałów (poza pierwszym „Wiadomości podstawowe") mamy „pomiary geodezyjne" lub „pomiary przemieszczeń". Nasuwa się więc pytanie: Czy to jest jeszcze geodezja, czy już geomatyka, a może geodezja to geomatyka albo jej część?

Dla mnie (przynajmniej do tej pory i na razie zdania nie zmieniam) **geodezja** była i jest dyscypliną naukową oraz techniczną, czyli zawodem, który można (ba! trzeba) wykonywać. Jako dyscyplina naukowa zajmuje się figurą Ziemi (m.in. jej kształtem, wymiarami), jako dyscyplina techniczna dostarcza informacji (danych) o położeniu, kształcie, geometrii różnych obiektów (nie tylko działek, lasów, pól itp.). Geodezja jest tym rodzajem działalności ludzkiej dostarczającej pierwiastka przestrzennego (a w zasadzie czasoprzestrzennego, bo znamy datę uzyskania tego pierwiastka przestrzennego), który następnie jest dołączany do różnych innych danych. Często też dane o położeniu, kształcie czy geometrii są nazywane danymi referencyjnymi czy georeferencyjnymi,



Rys. 2.1. Geoinformatyka jako nauka (Wilson J.P. i in., 2008)

czyli danymi, do których odwołują się inne dane, np. dane tematyczne. Dla systemu informacji o zanieczyszczeniach danymi tematycznymi będą dane o zanieczyszczeniach, a dane geodezyjne czy przestrzenne będą danymi referencyjnymi, czyli opisującymi dane tematyczne (inaczej mówiąc, danymi, do których dane tematyczne się odwołują). O wzajemnych relacjach pomiędzy geodezją, informatyką, naukami o Ziemi i geoinformatyką można też przeczytać w nowej publikacji „Podstawy budowy infrastruktury informacji przestrzennej" (Pachelski W. i in., 2012).

Kolejne pytanie brzmi: Czy geomatyka (lub jak niektórzy mówią geoinformatyka) jest odrębną dyscypliną nauki? Czy ma własne metody i język do prowadzenia badań? Prof. Gaździcki w swoim leksykonie rozdziela geoinformatykę od geomatyki (Gaździcki J., 2003). **Geomatyka** jest dyscypliną naukowo-techniczną, która zajmuje się pozyskiwaniem, interpretowaniem, upowszechnianiem i praktycznym wykorzystaniem geoinformacji. Natomiast **geoinformatyka** jest dyscypliną zajmującą się przetwarzaniem geoinformacji za pomocą komputerów, czyli jest częścią geomatyki.

Skłaniałbym się raczej w stronę uznania geomatyki za odrębną dyscyplinę naukową, chociażby z tego powodu, iż dostarcza nowej wiedzy (nawet gdyby przyjąć, że nie opracowała sobie właściwych metod badań i odpowiedniego języka). Rozumiem przez to, że poprzez nałożenie na szereg informacji aspektu czasowo-przestrzennego i przeprowadzenie analiz dowiemy się o nowych zależnościach między różnymi pojęciami (zjawiskami) znanymi dzisiaj. Te nowe zależności wniosą istotny wkład (tak zakładam) w bezpieczeństwo, zdrowie ludzi itp., a ich odkrycie będzie wynikiem przeprowadzenia analiz przestrzennych na zbiorach informacji (danych).

Do traktowania geoinformatyki jako dyscypliny nauki skłaniają się też autorzy licznych publikacji. W książce „The handbook of geographic information science" znajduje się rysunek (rys. 2.1) przedstawiający składniki geoinformatycznej nauki (Wilson J.P. i in., 2008). Można się spierać, czy rzeczywiście wszystkie zawarte w nim elementy są częściami geoinformatyki. Ale nawet jeśli uznamy, że nie, to i tak znajdują się „w okolicach" geoinformatyki: dostarczają danych, korzystają z wyników analiz itp. Autorzy wspomnianej książki starają się także wskazać dalsze kierunki rozwoju tej dyscypliny, przy czym wyraźnie zaznaczają, że są one skażone ich własnymi doświadczeniami i preferencjami.

Jeszcze inni dzielą geoinformatykę jako dyscyplinę nauki na „zwykłą" i „zaawansowaną", bo tak chyba można i należy odczytać tytuł książki „Advanced geoinformation science" (Yang Ch. i in., 2011). W niej także – oprócz rozdziałów o platformach komputerowych, analizach przestrzennych – jest rozdział o wizji geoinformatyki jako nauki.

„Świetlaną" przyszłość rysują przed geoinformatyką autorzy pozycji „Internet GIS. Distributed Geographic Information Services for the Internet and wireless network", w której są omawiane kwestie związków systemów informacji geograficznej (GIS – Geographic Information System) z internetem (Peng Z.-R. i in., 2003). Umożliwienie udostępniania danych przez internet, przeprowadzania analiz z wykorzystaniem usług sieciowych, prace związane np. z GIS-em dla urządzeń mobilnych to kierunki rozwoju geoinformatyki.

3. Modelowanie pojęciowe i schemat aplikacyjny

3.1. Modelowanie pojęciowe

Modelowanie pojęciowe (lub jak mówią inni – modelowanie obiektowe) jest stosunkowo nową metodą (techniką) modelowania. Jej początków należy szukać we wczesnych latach 80. XX w. Metody obiektowe polegają głównie na wyróżnieniu obiektów, które mogą łączyć w sobie możliwość przechowywania danych oraz wykonywania pewnych operacji (Jaszkiewicz A., 1997), czego brak był wadą dotychczasowych metod strukturalnych (podział systemu na składowe pasywne i aktywne).

Modelowanie jest rodzajem działalności ludzkiej, która ma doprowadzić do sformułowania modelu jakiegoś systemu. Model powinien być kompletny, jednoznaczny, niezależny od późniejszych implementacji, czyli języków, platform sprzętowych, środowisk. Modelowanie pojęciowe w kontekście informacji geograficznej oznacza skonstruowanie modelu systemu informacji geograficznej.

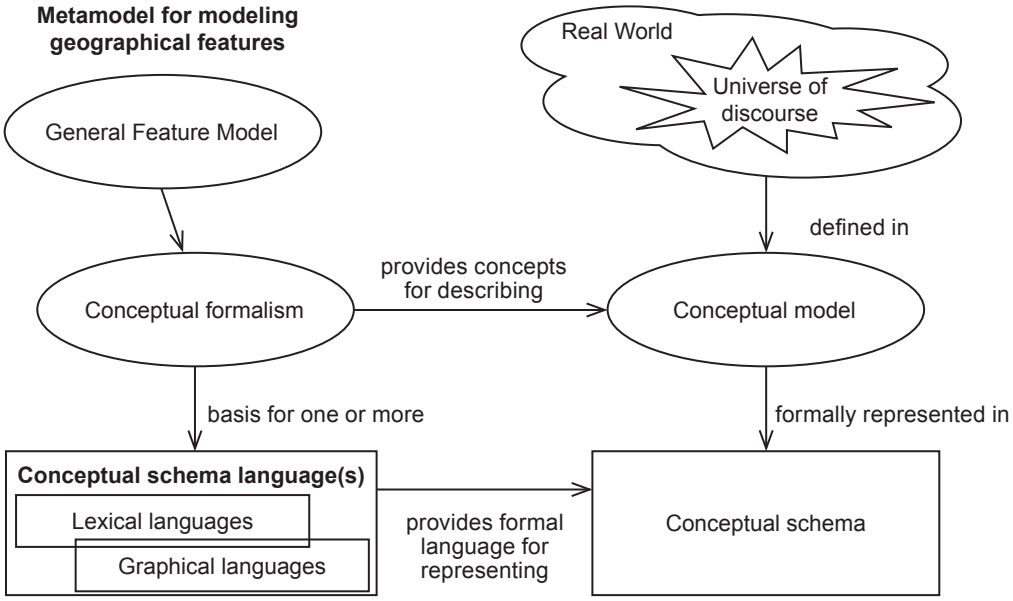
W normie ISO 19101 **modelowanie pojęciowe** (*conceptual modeling*) zdefiniowano jako proces tworzenia abstrakcyjnego opisu wybranego fragmentu rzeczywistości i zbioru powiązanych z nim pojęć (ISO 19101, 2002). Z podaniem przykładów elementów świata rzeczywistego, których opis powinien się znaleźć w modelu, nikt chyba nie będzie miał kłopotów: rzeki, miasta, działki, ludzie, ale też prawa do nieruchomości czy ograniczenia w tych prawach. Natomiast pojęciami, które są związane z elementami świata rzeczywistego i też powinny znaleźć się w modelu, są np. linie, punkty, powierzchnie, czyli według normy ISO 19101 „krąg geometrycznych konstrukcji". Konstrukcje te są wykorzystywane do opisu geometrii (kształtu, położenia) elementów modelu, czyli jednej z cech obiektów.

Powstanie modelowania pojęciowego łączy się z powstaniem technik (języków) programowania obiektowego (lub jak niektórzy je nazywają – obiektowych języków programowania). Podstawą metod obiektowych, jak sama nazwa wskazuje, jest wyodrębnienie ze świata obiektów, które są dla nas z jakiegoś punktu widzenia istotne i o których będziemy gromadzić dane. Można powiedzieć, że świat w dużej części składa się z obiektów i modelowanie pojęciowe ma dać opis zbliżony do tej części świata, którą modelujemy. Nie zajmujemy się późniejszą implementacją – interesują nas tylko obiekty, które łączą część statyczną (właściwości obiektu) z częścią dynamiczną (operacje, które mogą zostać wykonane na obiekcie).

Ten sposób podejścia pozwala na łatwy podział złożonego problemu na podproblemy (Pachelski W. i in., 2012; Schmuller J., 2003), którymi można się zajmować oddzielnie (jest to tzw. **zasada dekompozycji**). **Zasada abstrakcji** dotyczy eliminowania (o eliminowaniu, czyli wyborze części z całości, będzie dalej) lub ukrywania (**hermetyzacji**) mniej istotnych elementów („widoczność" atrybutów – zob. dodatek A). Innymi elementami modelowania pojęciowego, z którymi możecie Państwo się spotkać, są dziedziczenie i polimorfizm. **Dziedziczenie** zostało dokładnie omówione w rozdziale 4 (obiekt może dziedziczyć od innego obiektu wszystkie jego elementy). **Polimorfizm** dotyczy operacji (metod) i oznacza, że operacje w przypadku różnych obiektów mogą nazywać się tak samo, ale składać z różnych działań. Przykładem niech będzie operacja *boundary* (zob. rys. 5.3 – klasa *GM_Object*), w wyniku wykonania której ma zostać utworzona granica obiektu. Operację tę dziedziczą wszystkie klasy (tak w UML są nazywane reprezentacje obiektów) z normy PN-EN ISO 19107. Ale inaczej się wyznacza i czym innym są: granica linii, granica powierzchni czy granica bryły. W każdej z klas, które reprezentują linie, powierzchnie i bryły, występuje operacja *boundary* i w każdym przypadku inne działania mają zostać wykonane i inny produkt ma powstać. Ale zawsze ma to być granica obiektu.

3.2. Model pojęciowy

Zarysy pierwszej wersji modelu pojęciowego, który powstaje w wyniku modelowania pojęciowego, zawsze rodzą się w głowie osoby modelującej. Zaczyna się od wyboru fragmentu świata, który zostanie zamodelowany (rys. 3.1). Koniecznie należy otaczający nas świat ograniczyć terytorialnie i merytorycznie. Ograniczenie terytorialne oznacza podjęcie decyzji, jaki obszar będziemy modelować: gminę, kraj, kontynent czy jakieś pasmo górskie, które leży na obszarze kilku krajów (*universe of discourse* z rys. 3.1). Ograniczenie merytoryczne oznacza wybór typów obiektów, idei, faktów, procesów, które powinny się znaleźć w modelu (*universe of discourse*). Świat jest zbyt złożony, znajduje się na nim zbyt wiele różnych elementów, by próbować umieścić wszystkie w jednym modelu. Decydujemy się, że będziemy budowali model dla systemu danych topograficznych lub turystycznych, lub medycznych, lub jeszcze jakichś innych. Ze wszystkich elementów (obiektów i nie tylko) znajdujących się na danym obszarze i związanych z daną tematyką wybieramy te, których opis znajdzie się w modelu i potem w systemie.



Rys. 3.1. Od świata rzeczywistego do schematu pojęciowego (ISO 19101, 2002)

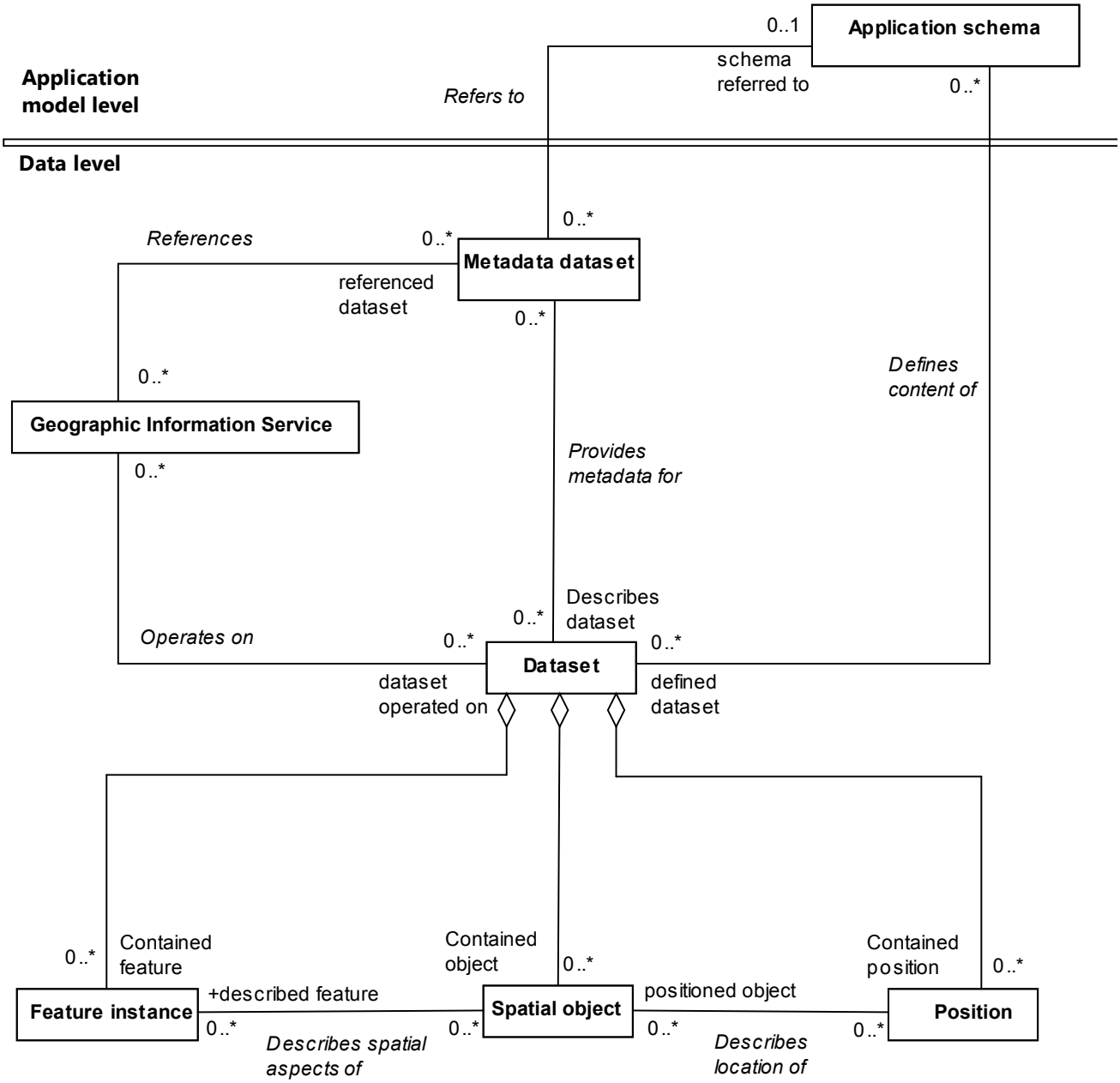
W następnym kroku zaczynamy się zastanawiać, o jakich właściwościach obiektów chcemy gromadzić informacje (duża część może nie być istotna, np. waga obiektów czy kolory). Zaczyna więc powstawać **model pojęciowy** (*conceptual model*). Jeśli konstruujemy model, który będzie opisywał więcej niż kilka obiektów, to pojawia się konieczność zapisania naszych pomysłów, by o nich nie zapomnieć, by móc komuś pokazać i podyskutować, by móc potem coś poprawić itp. Innymi słowy nakładamy na nasz model formalizm (zgodnie z rys. 3.1 *conceptual formalism* w postaci *conceptual schema language* – języka schematu pojęciowego) i model (wraz z kolejnymi wersjami) zostaje zapisany w wybranym języku formalnym służącym do zapisu modeli pojęciowych, np. UML. Otrzymujemy wtedy **schemat pojęciowy** – *conceptual schema*, czyli zapisany model pojęciowy. Schemat pojęciowy dla konkretnej dziedziny jest nazywany **schematem aplikacyjnym**.

3.3. Schemat aplikacyjny

W normie ISO 19101 jest zdefiniowany tzw. *domain reference model* (DRM), co można przetłumaczyć jako dziedzinowy model odniesienia, przy czym jako dziedzinę należy rozumieć informację geograficzną (ISO 19101, 2002). W zamierzeniach jest on przeznaczony dla wszystkich, którzy zajmują się lub są zainteresowani informacją geograficzną, i ma pomóc zrozumieć związki łączące schemat pojęciowy ze zbiorem danych. Wszystkie te zależności są zapisane z wykorzystaniem bardzo ogólnych pojęć.

Rys. 3.2 przedstawia *high-level view of the domain reference model*, co w luźnym tłumaczeniu można przyjąć jako spojrzenie z bardzo wysokiego poziomu na DRM. Rysunek został wyraźnie podzielony na dwa poziomy:

- *application model level* – poziom schematu aplikacyjnego,
- *data level* – poziom danych.



Rys. 3.2. Definicja *domain reference model* (ISO 19101, 2002)

O pewnych elementach występujących na rys. 3.2, takich jak cyfry czy gwiazdki, dowiecie się Państwo w rozdziale 4 o języku UML. Linie łączące prostokąty (które reprezentują pewne pojęcia, jak metadane zbioru – *metadata dataset* czy schemat aplikacyjny – *application schema*) oznaczają relacje (zależności) łączące te pojęcia.

Schemat aplikacyjny definiuje zawartość zbioru danych (chodzi tu o definicję typów obiektów, typów atrybutów, typów czy rodzajów relacji itp.), co jest pokazane na rys. 3.2 relacją łączącą schemat ze zbiorem danych.

Bardzo istotna jest dolna część rysunku, na której widnieje zawartość zbioru. Zbiór składa się z:

- *feature instance* – czyli konkretnych (instancje) obiektów opisanych atrybutami, operacjami (jeśli występują), asocjacjami z innymi obiektami;
- *spatial object* – czyli obiektów „przestrzennych", które opisują aspekty przestrzenne instancji obiektów, takimi obiektami są np. punkty, linie (krzywe, łamane), powierzchnie;
- *position* – opis pozycji w przestrzeni i czasie w jednostkach, które są zdefiniowane przez układ odniesienia (w definicji układu odniesienia znajdują się definicje osi i jednostek układu współrzędnych).

Na pierwszy rzut oka wydaje się, że coś jest nie tak, bo przecież położenie w przestrzeni powinien określać obiekt przestrzenny! Po cóż więc jeszcze jakiś element opisujący pozycję (*position*)? Otóż istnienie bezpośredniej pozycji jest uzależnione od istnienia przestrzennego obiektu, opisującego tę pozycję. Jeśli chcemy mieć określoną jakąś pozycję o współrzędnych X, Y, to musi istnieć obiekt przestrzenny – punkt o takich współrzędnych.

Na rys. 3.3 przedstawiono zawartość schematu aplikacyjnego – cały czas oczywiście używamy kategorii ogólnych. Tym razem przestrzeń rysunku jest podzielona na trzy części:

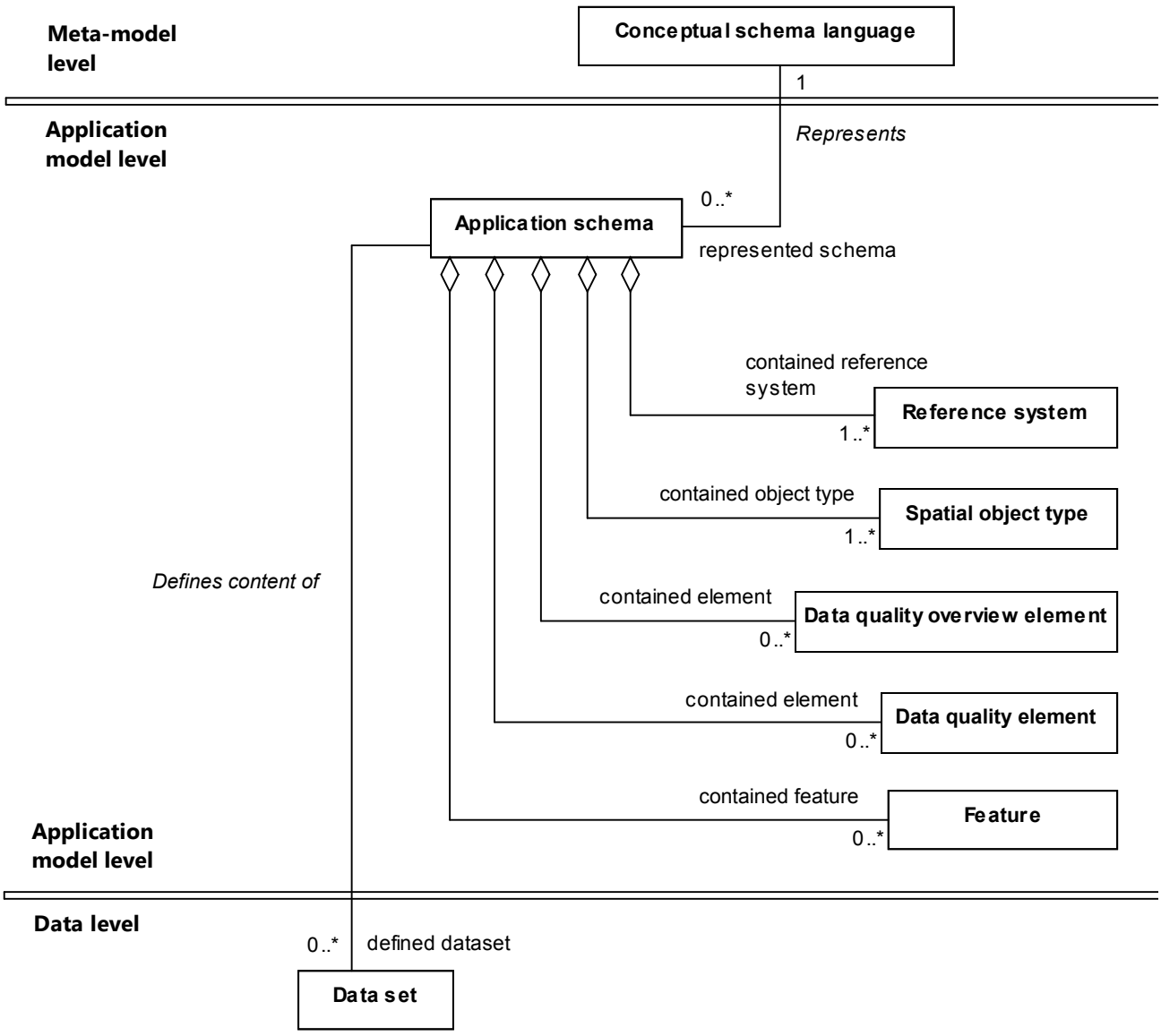
- *meta-model level* – poziom metamodelu,
- *application model level* – poziom schematu aplikacyjnego,
- *data level* – poziom danych.

Na poziomie metamodelu znajduje się definicja *conceptual schema language* – języka schematu pojęciowego. W naszym przypadku jest to UML, i to właśnie w UML-u będą zapisywane wszelkie schematy aplikacyjne lub ich fragmenty.

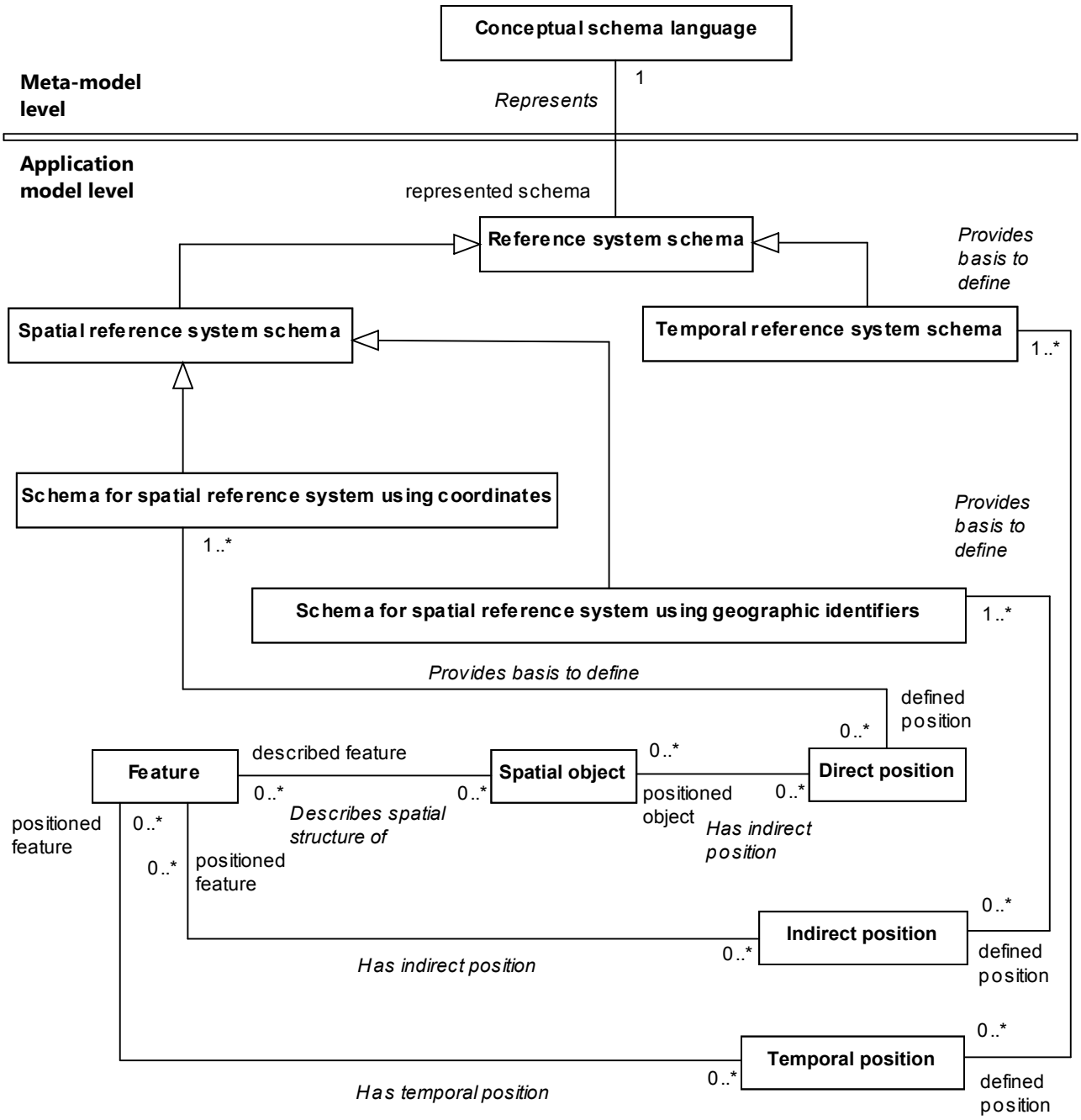
Poziom schematu aplikacyjnego wypełnia schemat aplikacyjny ze swoimi częściami składowymi, czyli (zaczynając od dołu):

- *feature* – reprezentacje obiektów z ich opisem atrybutowym, asocjacjami, ograniczeniami, operacjami itp.,
- *data quality element* – elementy jakości danych (są opisane w rozdziale o normach),
- *data quality overview element* – przeglądowe elementy jakości (one też są opisane w rozdziale o normach),
- *spatial object type* – typy obiektów przestrzennych, które będą definiować aspekty przestrzenne *feature*,
- *reference system* – system odniesienia, do którego są odnoszone informacje opisujące położenie i kształt obiektów w przestrzeni.

Na samym dole znajduje się poziom danych. Wszystkie elementy zdefiniowane w schemacie aplikacyjnym powinny się znaleźć w zbiorze danych. Jedynym wyjątkiem może być sytuacja, gdy na obszarze, dla którego są gromadzone dane, nie ma reprezentantów jakiegoś typu zdefiniowanego w schemacie (np. na danym obszarze nie będzie autostrady, jak w woj. podlaskim).



Rys. 3.3. Składniki schematu aplikacyjnego (ISO 19101, 2002)

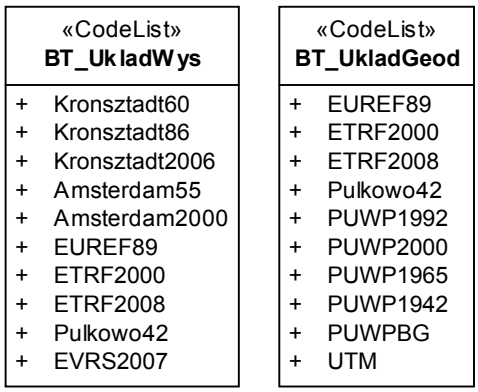


Rys. 3.4. Układ odniesienia w schemacie aplikacyjnym (ISO 19101, 2002)

Na rys. 3.4 są przedstawione szczegóły dotyczące układu odniesienia. Na górze rysunku znajduje się poziom metamodelu ze zdefiniowanym językiem schematu. Na poziomie schematu mamy przedstawiony (bardzo ogólnie) schemat układu odniesienia i sposób definiowania pozycji. Stosujemy najczęściej dwa układy odniesienia: przestrzenny i czasowy. Wśród przestrzennych wyróżnimy układ wykorzystujący współrzędne – *schema for spatial reference using coordinates*. Z jego pomocą można dokonać bezpośredniej definicji pozycji (*direct position*), która określa położenie obiektu przestrzennego (*spatial object*) służącego do opisania aspektów przestrzennych obiektów (*features*), o których będziemy gromadzić dane w bazie.

Drugim rodzajem przestrzennego układu odniesienia jest układ wykorzystujący identyfikatory geograficzne (np. adres) – *schema for spatial reference using geographic identifiers*. W tym przypadku definicja pozycji jest pośrednia (*indirect position*) i w ten sposób opisujemy położenie obiektów (*feature*) bez wykorzystania obiektów przestrzennych. Sposób został nazwany pośrednim, ponieważ, by pokazać, gdzie obiekt się znajduje, i tak trzeba przejść na współrzędne. Gdy położenie mam opisane adresem, muszę wiedzieć, gdzie leży np. plac Politechniki, by umieścić go we właściwym miejscu na mapie lub ekranie monitora. Używając identyfikatorów geograficznych, trzeba pamiętać o przejściu na współrzędne, przynajmniej w momencie ich obrazowania.

Do definiowania położenia na osi czasu dysponujemy jednym układem – *temporal reference system schema*, w którym jest określana pozycja na osi czasu (*temporal position*).



Rys. 3.5. Przestrzenne układy odniesienia w Polsce

Na rys. 3.5 pozwoliłem sobie wymienić nazwy wybranych tylko przestrzennych układów odniesienia, które obowiązywały lub obowiązują w Polsce (jak Czytelnicy być może zauważyli, w prostokącie z lewej strony są wymienione układy wysokościowe). O elementach, takich jak *«CodeList»*, „BT_", i wyjaśnienia, czym są te prostokąty, można przeczytać w rozdziale o języku UML. Na rysunku nie znalazły się nazwy wielu układów lokalnych.

A układ odniesienia dla określania pozycji w czasie mamy jeden!? Rzecz, przynajmniej dla mnie, zastanawiająca. Częściowo można zrozumieć potrzebę kilku (ilu?) przestrzennych układów odniesienia, ponieważ trochę inna będzie definicja takiego układu dla potrzeb przedstawienia Ziemi, a trochę inna dla mapy zasadniczej, czyli „wielkoskalowego opracowania kartograficznego zawierającego aktualne informacje o przestrzennym rozmieszczeniu obiektów ogólnogeograficznych oraz elementach ewidencji gruntów i budynków, a także sieci uzbrojenia terenu: nadziemnych, naziemnych i podziemnych" – jak o mapie zasadniczej mówi ustawa *Prawo geodezyjne i kartograficzne* (Pgik, 2010). Należy wyraźnie podkreślić, że model (czy schemat) opisuje, co system ma robić, ale nie decyduje, w jaki sposób (Schmuller J., 2003). Schemat aplikacyjny jest abstrakcyjnym opisem w kategoriach ogólnych, które nie są związane i nie decydują o późniejszej implementacji schematu.

Po tych kilku dywagacjach na temat schematu aplikacyjnego w kategoriach ogólnych przejdziemy do rzeczy bardziej konkretnych. W kolejnych krokach schemat powinien stawać się coraz bardziej szczegółowy, uzupełniany o powiązania między obiektami umieszczonymi w schemacie, o ograniczenia, liczności itp., czyli elementy, które zostaną dokładniej omówione w dalszej części książki, a które są niezbędne do budowy kompletnego schematu. Tabela 3.1 przedstawia elementy wchodzące w skład modelu pojęciowego oraz ich reprezentacje w UML.

Tabela 3.1. Elementy schematu pojęciowego

Fragment rzeczywistości	UML
Obiekty fizycznie istniejące lub nie, pojęcia, idee – elementy, które z jakiegoś powodu są dla nas istotne –> obiekty	Klasa
Właściwości obiektów	Atrybut
Połączenia pomiędzy obiektami	Asocjacja
Ograniczenia wpływające na funkcjonowanie obiektów	Ograniczenie
Działania, które mogą być wykonane na obiektach lub części ich właściwości	Operacja

W schemacie są jeszcze umieszczane dodatkowe informacje, np. nazwy ról asocjacji czy liczności – wszystkie są omówione w rozdziale 4.

Klasy połączone asocjacjami są zapisywane w postaci modeli (nazywanych schematami aplikacyjnymi). Jeśli klas jest dużo, to schemat aplikacyjny dzielimy na części z bardzo prostego powodu – nie jesteśmy w stanie objąć rozumem (bo wzrokiem zapewne tak) zbyt wiele klas naraz. Taki podział na mniejsze, logicznie powiązane części ułatwia sprawę. Są one zapisywane w pakietach posiadających swoje nazwy i zawierających klasy, które autor modelu postanowił umieścić w jednym pakiecie.

Ta książka nie jest podręcznikiem modelowania pojęciowego, dlatego po więcej informacji o modelowaniu odsyłam do bogatej literatury przedmiotu.

4. Język UML

Język modelowania pojęciowego (*Unified Modeling Language*) jest językiem formalnym. Jeśli coś w nim zapiszę, to każdy odczyta dokładnie to, co napisałem (nawet jeśli będzie miał nieco złej woli). Innymi słowy, jest to próba uwolnienia się od problemów związanych z niejednoznacznością języków naturalnych, w których tę samą wypowiedź można różnie zrozumieć. W konsekwencji każdy element UML znaczy coś konkretnego i tylko to – nic więcej. I każdy element może zostać wykorzystany tylko w określonych sytuacjach.

UML stworzyli: Grady Booch, Ivar Jacobson i James Rumbaugh, którzy byli pracownikami Rational Software Corporation. UML (według Rational) jest językiem przeznaczonym do specyfikowania, konsultowania, wizualizacji i dokumentowania elementów systemów. Jest też ugruntowanym językiem w środowisku inżynierii oprogramowania (Śmiałek M., 2005).

UML jest podzielony na diagramy, każdy z diagramów składa się z elementów, które służą do przedstawienia różnych sytuacji. Na przykład do przedstawienia dynamiki systemu, procesu służą diagramy czynności; do przedstawienia funkcjonalności jakiegoś systemu służy diagram przypadków użycia. W tej książce UML został ograniczony tylko do jednego diagramu: **diagramu klas** (lub inaczej: **diagramu struktury statycznej**). Ten diagram jest wykorzystywany m.in. do opisu informacji przestrzennej. Dyrektywa INSPIRE (Dyrektywa INSPIRE, 2007) i różne dokumenty z nią związane wykorzystują „oryginalny" UML nieco zmodyfikowany, by lepiej spełniał swoje zadania. Jest on zdefiniowany w specyfikacji technicznej ISO 19103 (ISO/TS 19103, 2005).

W książce tej będę opisywał UML „dostosowany" do INSPIRE, czyli ten zdefiniowany w ISO/TS 19103.

W niektórych opracowaniach są wyróżniane perspektywy UML: pojęciowa, specyfikacyjna, implementacyjna (Fowler M. i in., 2002). Dwie ostatnie dotyczą głównie modeli oprogramowania. My ograniczymy się do perspektywy pojęciowej, a więc konstruowane modele będą dotyczyły pojęć danej dziedziny (Fowler M. i in., 2002). W naszym przypadku będą to modele (schematy aplikacyjne), które mają być niezależne od późniejszego sposobu implementacji (rodzaju bazy danych, aplikacji, systemu operacyjnego, sprzętu komputerowego itp.). Modelujemy tylko interesujący nas fragment rzeczywistości.

Jak każdy język, UML ma swój alfabet dostosowany do potrzeb, które język ma spełniać – służyć do zapisu modelu. Zaczniemy więc od alfabetu UML. „Wypowiedziami" w UML są schematy aplikacyjne. Składają się one z różnych elementów: pakietów, klas, atrybutów, relacji, ograniczeń, liczności, stereotypów, typów itp. O każdym z tych elementów poniżej trochę opowiem. Wszystkie je zaliczam do alfabetu.

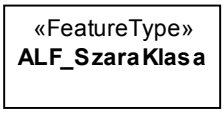
4.1. Alfabet

Klasy

Klasa jest abstrakcyjnym przedstawieniem obiektu istniejącego lub nie; obiektu, który z jakichś względów jest dla nas ważny i chcemy go umieścić w modelu. Z drugiej strony taki obiekt umieszczony w modelu jest reprezentantem wszystkich obiektów tego typu, które są tak samo opisywane (czyli dla których zbieramy informacje o takich samych właściwościach/cechach). Reasumując, można stwierdzić, że:

obiekt => klasa => wszystkie obiekty tego samego rodzaju i tak samo opisywane.

Klasa jest przedstawiana na modelu w postaci prostokąta, który może być podzielony na kilka części. Na rys. 4.1 klasa jest przedstawiona w sposób ogólny. Wynika z niego, że SzaraKlasa jest klasą, że należy do modelu o skrócie ALF, ma stereotyp «*FeatureType*» i albo nie ma atrybutów, albo je ma, ale ich nie przedstawiono. Ten ogólny sposób przedstawienia klasy stosuje się w przypadku, gdy klas jest bardzo dużo i po prostu nie ma miejsca na dokładniejszy opis albo gdy specjalnie zależy nam tylko na pokazaniu, jakie klasy będą występować w modelu i jakie zależności (relacje) będą je łączyć.



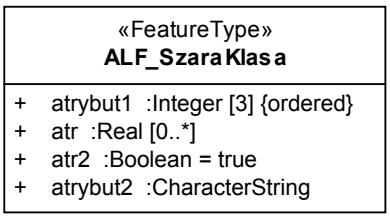
Rys. 4.1. Klasa – sposób 1

Stereotyp (a w zasadzie jego nazwa) jest umieszczany w podwójnych nawiasach (niektórzy nazywają je kątowymi) nad nazwą klasy. Stereotypy rozszerzają notację UML, tzn. dzięki ich wykorzystaniu możemy rozszerzyć podawane informacje dotyczące klasy. W tym przypadku sygnalizują, jaką rolę dana klasa odgrywa w modelu. Stereotyp «*FeatureType*» jest używany głównie do klas opisujących obiekty przestrzenne, «*DataType*» do klas definiujących typy danych, a «*CodeList*» i «*Enumeration*» do list wyliczeniowych. W dalszej części powiem jeszcze o stereotypach «*Union*» i «*Voidable*».

Już wiemy, że nazwą klasy na rys. 4.1 jest „SzaraKlasa". Przedrostek „ALF" informuje, do jakiego modelu dana klasa należy. Jest to istotne, zwłaszcza gdy będziemy w konstruowanym schemacie aplikacyjnym wykorzystywać klasy z różnych modeli. Taki sposób zapisu nazw klas (z przedrostkami) został przyjęty m.in. w INSPIRE. „ALF" będzie u nas oznaczał alfabet.

Atrybuty

Bardziej szczegółowy sposób definiowania klasy w schemacie jest przedstawiony na rys. 4.2. Gdy decydujemy, że jakiś obiekt ma się znaleźć w bazie danych, to zależy nam na umieszczeniu jakichś danych o tym obiekcie. Zależy nam, by opisać pewne jego cechy, które nas interesują. W UML cechy czy właściwości obiektu opisuje się z wykorzystaniem atrybutów klasy (która reprezentuje obiekt w modelu). Na rys. 4.2 są zdefiniowane cztery atrybuty.



Rys. 4.2. Klasa – sposób 2

Atrybut jest opisem właściwości obiektu. Jego definicja powinna się składać z określenia widoczności, po niej musi się pojawić nazwa atrybutu, po dwukropku powinien zostać podany typ wartości, jakie może przyjąć, liczność (czyli ile wartości ma lub może przyjąć). To są elementy konieczne. Opcjonalnie w skład definicji mogą jeszcze wchodzić różne informacje, np. czy wartości mają być uszeregowane (*{ordered}*), czy ustalamy także wartość domyślną. Autorzy UML podają ogólną definicję atrybutu (Booch G. i in., 2001):

[widoczność] nazwa [liczebność] [:typ] [=wartość początkowa][określenie właściwości]

Poniżej zostaną krótko omówione elementy składające się na definicję atrybutu. Można założyć, że elementy w nawiasach są umieszczane w definicji atrybutu w modelu w zależności od ogólności opisu. Nazwa musi być podana zawsze z wyjątkiem przypadków, gdy opis jest tak ogólny, że nie ma wymienianych atrybutów. Liczebność jest też często nazywana licznością i oznacza dokładnie to samo.

Przed nazwą każdego z atrybutów znajduje się plus („+"), który określa widoczność atrybutu. Plus oznacza, że atrybut jest publiczny (*public*), czyli jest widoczny (zob. dodatek A, gdzie są podane inne typy widoczności). Tylko takimi atrybutami będziemy się zajmować. Kwestie dotyczące atrybutów prywatnych, chronionych na naszym etapie poznawania UML-a nie są niezbędne, a mogą być wręcz szkodliwe!

- Atrybut pierwszy – nazwą jest „atrybut1", typem *Integer*, czyli liczby całkowite. Liczność – trzy wartości tego atrybutu mają zostać zapisane i mają być one uporządkowane (jeśli „atrybut1" będzie oznaczał współrzędne jednego punktu z pomiaru GPS, to ich kolejność ma zostać ustalona i zawsze pozostawać taka sama, np. X, Y, Z).
- Atrybut drugi – nazwą jest „atr", typem *Real*, czyli liczby rzeczywiste. Liczność może być różna i zmieniać się od 0 do wartości niezdefiniowanej – *.
- Atrybut trzeci – nazwą jest „atr2", typem *Boolean*, czyli typ logiczny (zob. w dodatku A). Wartością domyślną jest „true". Liczność wynosi jeden i taka jest standardowo przyjmowana, dlatego jedynek nie umieszcza się w opisie atrybutów. Liczność wynosząca 1 oznacza, że atrybut jest „**wymagalny**", czyli

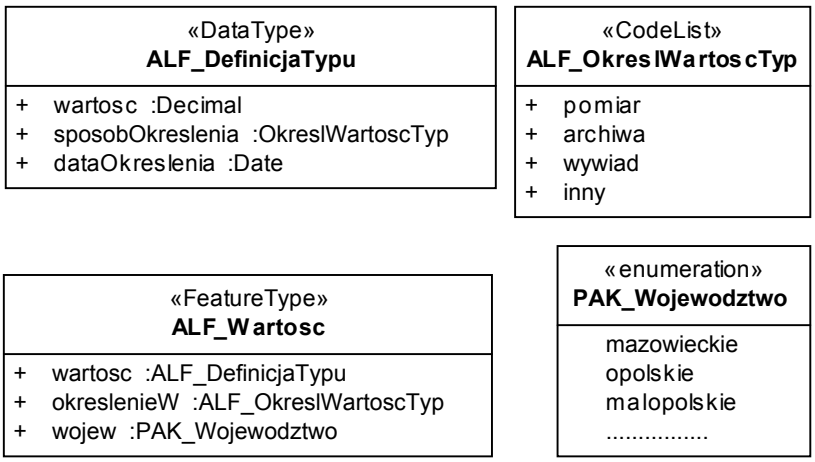
w późniejszej bazie danych „atr2" (i „atrybut2") muszą zostać wypełnione wartością, w przeciwnym razie nie powinien zostać zapisany w bazie cały obiekt (z powodu braku jednego atrybutu!).

- Atrybut czwarty – nazwą jest „atrybut2", typem *CharacterString*, czyli ciąg znaków (tekst).

W powyższych przykładach zostały wykorzystane tzw. typy proste. Występują też (bo muszą) typy złożone (zob. dodatek A): implementacyjne, zbiorowe, rekordowe itp. Wzmianki o niektórych z nich będą w dalszej części książki. Typ atrybutu może zostać także zdefiniowany w postaci klasy o odpowiedniej strukturze i stereotypie.

Liczność jest parametrem wchodzącym w skład definicji atrybutu i relacji. Podajemy ją w nawiasach kwadratowych dla atrybutów lub bez nawiasów dla relacji. W przypadku relacji należy podać dwie liczności – dla każdego jej zakończenia, bo relacja łączy dwie klasy (czasami na modelu można tak manipulować znakiem graficznym danej relacji, że wygląda to tak, jakby za pomocą jednej relacji połączono kilka różnych klas; bardzo często tak się robi w przypadku, gdy od jednej klasy dziedziczy kilka klas, ale to tylko tak wygląda na rysunku). Liczności równej 1 najczęściej na modelu się nie umieszcza – przyjmuje się, że jest to wartość standardowa. Atrybuty o liczność 1 są nazywane „wymagalnymi", czyli muszą zostać wypełnione wartością pod rygorem nieutworzenia obiektu w bazie danych. Podobnie jest w przypadku relacji.

Dla atrybutów liczność większa od 1 oznacza, że atrybut taki jest wielowartościowy – na rys. 4.2 „atrybut1" ma zawierać (można w nim zapisać) trzy wartości. Dla relacji liczność oznacza liczbę instancji klasy (czyli obiektów z bazy danych), które biorą jednocześnie udział w relacji. Często liczność jest podawana jako przedział, np. 1..*, 0..1, 3..15, gdy liczba wartości zapisanych w atrybucie lub liczba obiektów w relacji może być różna, ale w ściśle określonych granicach. Niektóre aplikacje umieszczają na modelu liczność po typie atrybutu, a nie przed nim.



Rys. 4.3. Klasa – sposób 2 cd.

Na rys. 4.3 przedstawiono klasę „ALF_Wartosc" z typami atrybutów, które są zdefiniowane w postaci klas. Klasa, która została utworzona po to, by zdefiniować nowy typ danych (na rys. klasa „ALF_DefinicjaTypu"), otrzymuje stereotyp «*DataType*». Klasa taka (poza nielicznymi wyjątkami) nie wchodzi w żadne relacje z innymi klasami.

Zdarza się, że atrybut może przyjąć wartości ze ściśle zdefiniowanej listy. Taki typ też jest definiowany jako klasa, której atrybutami są wartości, jakie może przyjąć atrybut. Klasa definiująca listę jest nazywana **klasą wyliczeniową** i otrzymuje stereotyp «*CodeList*» (klasa „ALF_OkreslWartoscTyp") lub «*Enumeration*» (klasa „PAK_Wojewodztwo"). «*CodeList*» występuje, gdy użytkownik pracujący z bazą danych może samodzielnie listę rozszerzyć, «*Enumeration*» otrzymują klasy, dla których lista zdefiniowana w schemacie aplikacyjnym jest ostateczna i potem – podczas działania aplikacji – nikt jej zmienić nie może. Należy wyraźnie zaznaczyć, że chociaż stereotyp «*CodeList*» jest często wykorzystywany w schematach będących częściami (zwykle załącznikami) rozporządzeń, to w tych przypadkach możliwość dodawania nowych elementów jest nie do zrealizowania, ponieważ zmiana czegokolwiek w opublikowanym rozporządzeniu wymaga jego nowelizacji.

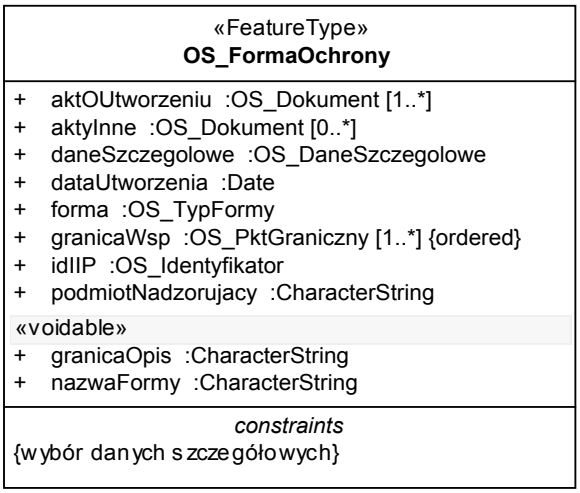
Przedrostek „PAK" w klasie „PAK_Wojewodztwo" oznacza, że klasa ta została zdefiniowana w modelu „PAK" – Podział Administracyjny Kraju. Z kolei typ atrybutu „dataOkreslenia" – „*Date*" jest zdefiniowany w specyfikacji ISO 19103 (ISO/TS 19103, 2005).

Nazwy klas, stereotypów, atrybutów, typów danych itp. też są elementami alfabetu. **Nazwa** jest ciągiem znaków (tak samo jak w językach naturalnych). Przy podawaniu nazw należy stosować kilka prostych zasad. Rozsądne jest oczekiwanie, by nazwa niosła ze sobą informację, co dana klasa, atrybut itp. reprezentują, jaką rolę odgrywają. Nazwy atrybutów z rysunku 4.2 nie są najlepsze, ale formalnie (poza tym, że są mało zrozumiałe) nic im zarzucić nie można. Nazwy powinny być rzeczownikami w liczbie pojedynczej. Gdy chcemy, by nazwa

składała się z kilku wyrazów, można je oddzielić myślnikiem, podkreśleniem lub następny wyraz zacząć pisać wielką literą, jak np. atrybut „okreslenieW" z rys. 4.3 czy typ tego atrybutu „OkreslWartoscTyp". Nie wolno w nazwach używać spacji! Nazwy klas poprzedzamy przedrostkiem (skrót nazwy modelu) i piszemy wielką literą. Nazwę atrybutu piszemy małą literą (widać to na dotychczasowych rysunkach). Nazwę stereotypów piszemy wielką lub małą literą. By uniknąć problemów ze stronami kodowymi w przypadku stosowania polskich znaków (co się zdarzało), najlepiej z nich zrezygnować i pisać np. „Slask" zamiast „Śląsk".

Ograniczenia

Na pewne elementy modelu można nakładać ograniczenia. Bardzo częste jest uzależnianie wartości czy zakresu wartości, jakie może przyjąć atrybut, od wartości innego atrybutu. Na rys. 4.4 jest pokazana klasa ze zdefiniowanym ograniczeniem. Dotyczy ono wyboru wartości atrybutu „daneSzczegolowe". Klasa „OS_FormaOchrony" została zdefiniowana w schemacie aplikacyjnym wykonanym dla Generalnej Dyrekcji Ochrony Środowiska i w sposób ogólny opisuje chronione formy.



Rys. 4.4. Klasa ze zdefiniowanym ograniczeniem

Opis bardziej szczegółowy znajduje się w atrybucie „daneSzczegolowe". Inaczej jest opisywany park krajobrazowy, park narodowy, użytek ekologiczny czy pomnik przyrody. Jaką formę będzie w bazie danych reprezentować klasa „OS_FormaOchrony", decyduje wartość atrybutu „forma". Gdy „forma" przyjmie wartość „obszarNatura2000", to dane szczegółowe muszą opisywać obszary Natura 2000. O ograniczeniach jest wzmianka w dodatku B. Ograniczenia mogą też dotyczyć liczności, np. atrybut ma liczność [0..1], ale w pewnym przypadku (po spełnieniu jakiegoś warunku) ma być atrybutem wymagalnym, czyli mieć liczność 1. Ograniczenia dotyczą działania aplikacji, są pewnym dodatkiem, więc dłużej nie będę się tutaj nad nimi rozwodził.

Stereotyp «Voidable»

Na rys. 4.4 pojawił się nowy stereotyp. «**Voidable**» jest rodzajem stereotypu, który nadajemy atrybutom lub relacjom. Ma on specjalne znaczenie i jest rozszerzeniem profilu UML zdefiniowanym w specyfikacji ISO/TS 19103. Atrybuty mające liczność 1 są tzw. atrybutami wymagalnymi. Ale może się tak zdarzyć, że nie będziemy mogli zdobyć informacji o wartości takiego atrybutu. Z drugiej strony chcemy móc zapisać taki obiekt w bazie danych. W takim przypadku trzeba zastosować specjalny atrybut, w którym zamiast wartości atrybutu zostanie zapisana informacja o przyczynach braku wartości. Oznaczeniem takiej sytuacji jest stereotyp «*Voidable*». Tym

Tabela 4.1. Wartości specjalnego atrybutu nilReason

Wartość	Definicja	Wartość w ISO 19136
Nie stosuje się	Nie ma zastosowania (wartości) w danym kontekście	*inapplicable*
Brak danych	Prawidłowa wartość atrybutu nie jest obecnie znana, ale właściwa wartość może nie istnieć	*missing*
Tymczasowy brak danych	Wartość atrybutu będzie dostępna w późniejszym terminie	*template*
Nieznany	Prawidłowa wartość atrybutu nie jest znana, ale właściwa wartość prawdopodobnie istnieje	*unknown*
Zastrzeżony	Wartość atrybutu jest zastrzeżona	*withheld*

specjalnym atrybutem jest *gml:nilReason*, który jest zdefiniowany w normie ISO 19136 (ISO 19136, 2007). Wartości tego atrybutu są przedstawione w tabeli 4.1.

Może się wydawać dziwne, jak dla atrybutu typu np. *Integer* zamiast wartości całkowitej można wpisać np. „*missing*". By móc zaimplementować taką sytuację, w normie ISO 19136 (ISO 19136, 2007) zostały zdefiniowane typy:

- *gml:booleanOrNilReason*,
- *gml:doubleOrNilReason*,
- *gml:integerOrNilReason*,
- *gml:NameOrNilReason*,
- *gml:stringOrNilReason*,

które umożliwiają wpisanie albo wartości zgodnej z danym typem lub wartości atrybutu specjalnego.

Operacje

Kolejna część prostokąta stanowiącego klasę może charakteryzować operacje. Operacje są usługami (inaczej procesami), które dana klasa (a w zasadzie jej implementacje w bazie danych, czyli konkretne obiekty) ma wykonać na polecenie innej klasy (rys. 4.5). Definicja operacji wymaga oczywiście podania jej nazwy (zaczynając od małej litery i dalej zgodnie z określonymi wcześniej regułami podawania nazw) i typu wyniku. Opcjonalnie można w nawiasach podać typy parametrów. Zapis jest w zasadzie identyczny jak w językach programowania, np. *Delphi*, podaje się definicje procedur. Diagram klas jest opisem, służy do zamodelowania struktury statycznej, więc operacje w modelach często nie występują. Tutaj podaję ten bardzo krótki opis dla porządku, że takie elementy też mogą wchodzić do definicji klasy.

Rys. 4.5. Klasa ze zdefiniowaną operacją

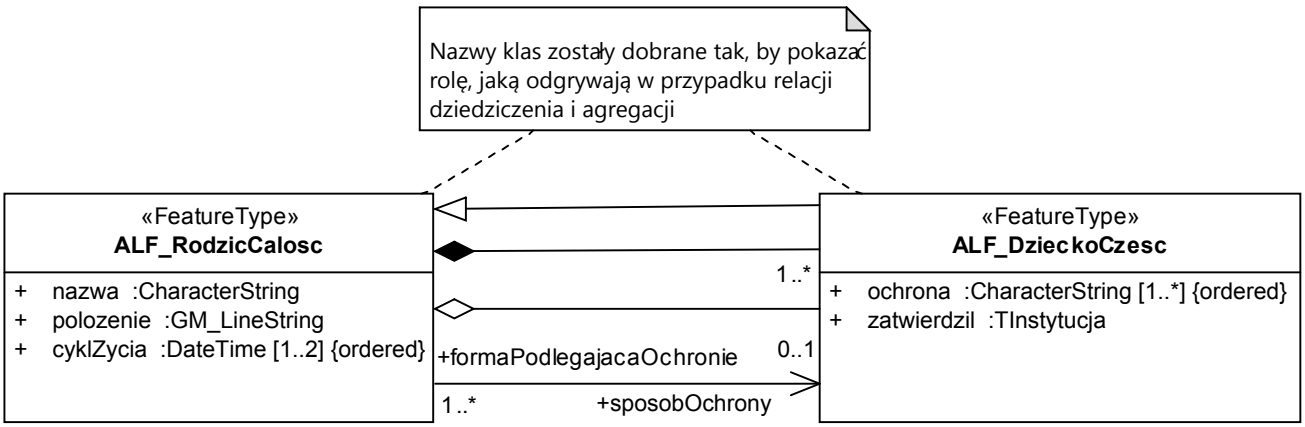
Do modelu można dodawać **notatki**, których głównym zadaniem jest wyjaśnienie dokładniej czegoś, co według autora modelu może wymagać dodatkowego komentarza (rys. 4.6). Mają wygląd prostokąta z zagiętym jednym rogiem. Notatki mogą być połączone z jakąś klasą (klasami), jak to jest na rys. 4.6, co ma pokazać, do czego odnosi się komentarz, lub mogą zostać umieszczone oddzielnie.

Relacje

Model, w którym zostały zdefiniowane tylko typy obiektów, bez podania związków zachodzących między nimi, nie jest modelem bazy danych. Jest raczej zbiorem definicji pojedynczych obiektów. Bazę danych tworzą obiekty połączone zachodzącymi między nimi relacjami oraz system zarządzania tą bazą. Na rys. 4.6 są przedstawione cztery najczęściej stosowane relacje. Pierwszą z nich, idąc od góry, jest relacja **dziedziczenia**, która oznacza, że klasa „ALF_DzieckoCzesc" przejmuje wszystkie atrybuty, relacje, ograniczenia itp. od klasy „ALF_RodzicCalosc", która jest wskazywana grotem strzałki. Przejmuje tzn., że oprócz „swoich" dwóch atrybutów ma także trzy atrybuty klasy „ALF_RodzicCalosc". Gdyby klasa „ALF_RodzicCalosc" była połączona relacją z jakąś inną klasą (np. ALF_X), to dzięki relacji dziedziczenia klasa „ALF_DzieckoCzesc" też byłaby połączona tą samą relacją z ALF_X. To właśnie przejmowanie cech innej klasy było powodem nazwania tej relacji „dziedziczeniem" w analogii do ludzi, którzy dziedziczą (przejmują) cechy rodziców. Istotna różnica między ludźmi a UML jest taka, że w UML klasa odgrywająca rolę „dziecka" przejmuje wszystkie cechy klasy będącej „rodzicem". Ludzkie dzieci zwykle dziedziczą część cech (często nie te, które rodzice uważają, że powinny).

Drugą relacją jest **agregacja silna** lub – jak jest często nazywana – **kompozycja**. Agregacja jest relacją całość – część i całością jest klasa znajdująca się po stronie zaczernionego rombu. Po stronie klasy będącej częścią „ALF_DzieckoCzesc" jest podana liczność [1..*], która oznacza, że w skład całości musi wchodzić przynajmniej jedna część, a może ich być wiele.

Trzecią relacją jest **agregacja słaba** lub po prostu **agregacja**. Całością, jak w przypadku kompozycji, jest klasa po stronie rombu. Można też podać liczność. Dla relacji, podobnie jak dla atrybutów, jeśli liczność nie jest podana, to oznacza, że wynosi ona 1.



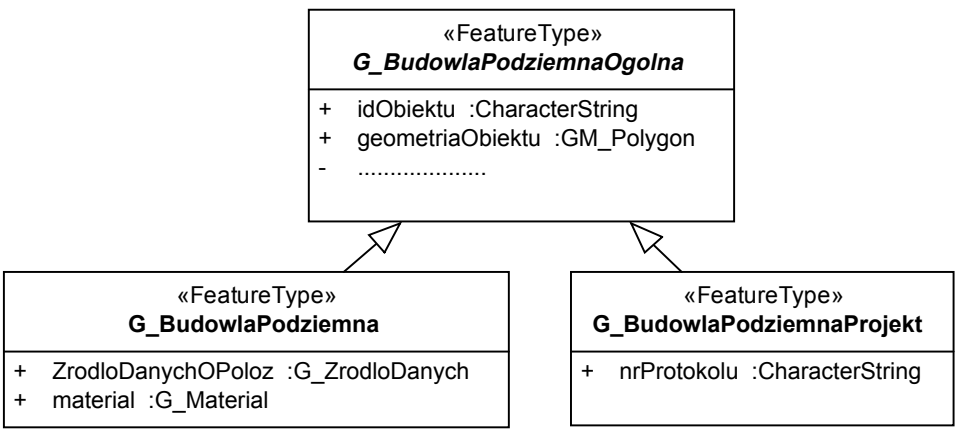
Rys. 4.6. Relacje łączące klasy

Zarówno relacja druga, jak i trzecia są agregacjami. Różnica między nimi polega na tym, że gdy z bazy danych usuwamy obiekt będący całością, to w przypadku kompozycji musimy też usunąć wszystkie jego części. Nie ma takiego obowiązku dla agregacji słabej. W takiej sytuacji klasa będąca częścią staje się klasą „samodzielną".

Ostatnia jest relacja **asocjacji**, która oznacza tylko, że obie klasy są ze sobą połączone jakimś związkiem. Istotną kwestią jest umieszczenie w modelu informacji, czym to połączenie jest. Służą do tego **opisy ról**, które informują, jaką rolę w danej asocjacji odgrywa dana klasa. Na rys. 4.6 z jednej strony jest forma (chociaż nazwa „ALF_RodzicCalosc" niezbyt na to wskazuje), która podlega ochronie, z drugiej mamy zdefiniowany w postaci klasy sposób ochrony. Określone też są liczności. Nie każda forma musi mieć przyporządkowany sposób ochrony [0..1]. Jednym sposobem ochrony musi zostać objęta co najmniej jedna forma [1..*]. Dzięki opisom ról i podaniu liczności „tajemnicza" asocjacja staje się relacją w pełni zrozumiałą. Ostatnią sprawą wymagającą wyjaśnienia jest grot strzałki pojawiający się przy klasie „ALF_DzieckoCzesc". Jest to pokazanie tzw. **nawigacji**, czyli wskazanie, skąd płynie informacja. W asocjacji z rys. 4.6 informacja o sposobie ochrony jest przekazywana do klasy „ALF_RodzicCalosc". Klasa może też zostać połączona relacją z samą sobą (rys. 4.8).

Na rys. 4.7 pojawił się nowy element, tzw. **klasa abstrakcyjna** (jej nazwę piszemy kursywą – jest to oznaczenie klas abstrakcyjnych). Kogoś mogłaby zdziwić taka klasa, ponieważ każda klasa jest przecież abstrakcyjnym przedstawieniem iluś obiektów. Tak, to prawda, ale wśród tych abstrakcji wyróżniamy jeszcze inne, tak jakby „bardziej zanurzone w abstrakcji". Klasa abstrakcyjna w schemacie UML przedstawia sobą element, który nie znajdzie się w bazie danych (nie będzie obiektów opisywanych taką klasą). Klasę abstrakcyjną umieszcza się w modelu w zasadzie w jednym celu – dziedziczenia. Inne klasy mają dziedziczyć od niej jej atrybuty, relacje, ograniczenia itp.

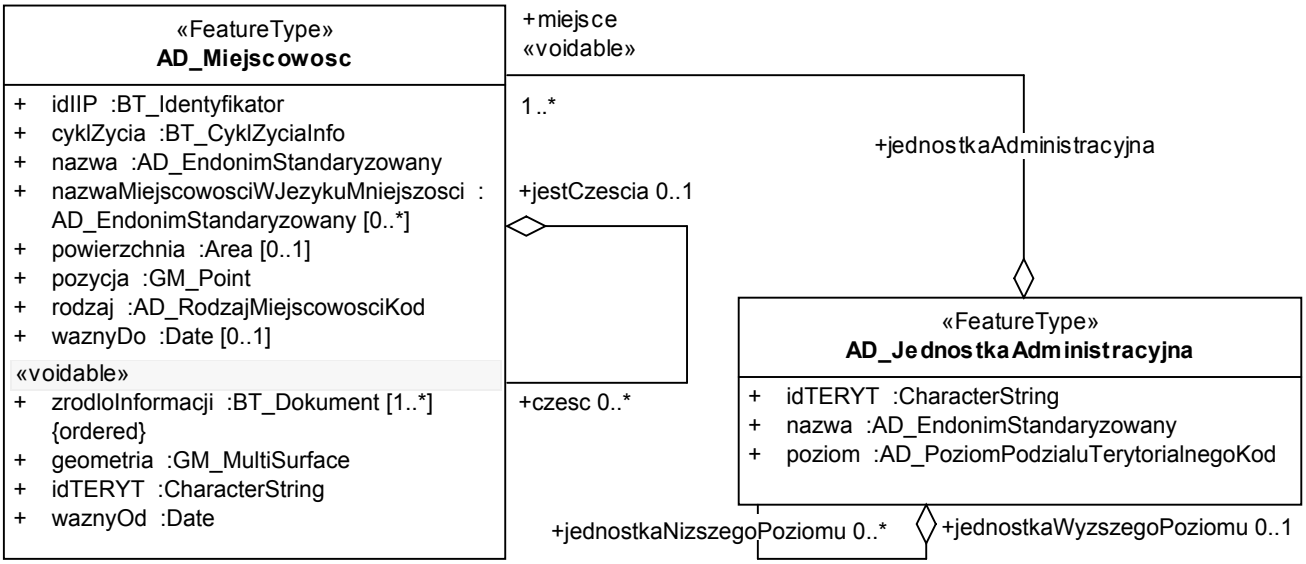
Rys. 4.7 jest fragmentem modelu umieszczonego w artykule opublikowanym w „Rocznikach Geomatyki" (Pachelski W. i in., 2007). Obiekty „budowla podziemna" i „budowla podziemna projektowana" były opisywane ośmioma takimi samymi atrybutami. Były możliwe dwie drogi. Można było umieścić w modelu dwie klasy: „G_BudowlaPodziemna" (G – tu oznacza geodezję) z dziesięcioma atrybutami i „G_BudowlaPodziemnaProjekt" z dziewięcioma atrybutami. Inny sposób jest przedstawiony na rys. 4.7, gdzie umieszczono trzy



Rys. 4.7. Dziedziczenie od klasy abstrakcyjnej

klasy: klasę abstrakcyjną „G_BudowlaPodziemnaOgolna", która będzie opisywana ośmioma wspólnymi atrybutami dla budowli podziemnej i budowli podziemnej projektowanej, dwie klasy: „G_BudowlaPodziemna" z dwoma atrybutami (które nie są wspólne) i „G_BudowlaPodziemnaProjekt" z jednym atrybutem (który nie jest wspólny) oraz relację dziedziczenia od klas „dzieci" do „rodzica", czyli klasy abstrakcyjnej. Dziedziczenie od „rodzica" oznacza, że w bazie danych obiekty, które w modelu są reprezentowane przez klasę „G_BudowlaPodziemna", będą charakteryzowane dziesięcioma atrybutami (ośmioma odziedziczonymi od „G_BudowlaPodziemnaOgolna" oraz dwoma „swoimi"). Podobnie będzie z budowlami projektowanymi, tylko będą miały dziewięć atrybutów.

Zamykając kwestie klas abstrakcyjnych, można stwierdzić, że są „zbiornikiem" atrybutów, relacji, ograniczeń i operacji, które są wspólne dla iluś klas. Jeśli w modelu jest klasa abstrakcyjna, to musi od niej dziedziczyć inna klasa (która też może być abstrakcyjna).



Rys. 4.8. Relacje agregacji (EMUiA, 2012)

Na rys. 4.8 jest podany przykład trzech relacji dla dwóch klas. Jest to fragment schematu aplikacyjnego z rozporządzenia o ewidencji miejscowości, ulic i adresów (EMUiA, 2012). Miejscowość leży (a przynajmniej powinna) na terenie jakiejś jednostki administracyjnej – stąd relacja agregacji pomiędzy tymi dwiema klasami. Klasa „AD_Miejscowosc" reprezentuje wszystkie miejscowości, także takie, które z jednej strony są odrębnymi miejscowościami, a z drugiej są częścią innej (należy się domyślać, że większej) miejscowości – stąd relacja agregacji klasy do siebie. By ta relacja była zrozumiała, są w niej opisane role:

- „jestCzescia" reprezentuje miejscowość nadrzędną, której częścią jest miejscowość podrzędna; liczność wynosi [0..1], ponieważ nie musi istnieć miejscowość nadrzędna, ale jeśli istnieje, to miejscowość podrzędna może być częścią tylko jednej miejscowości nadrzędnej;
- „czesc" reprezentuje miejscowość podrzędną, która stanowi część nadrzędnej; liczność wynosi [0..*], ponieważ miejscowość nie musi być podrzędna, ale jednocześnie do miejscowości nadrzędnej może należeć kilka miejscowości podrzędnych.

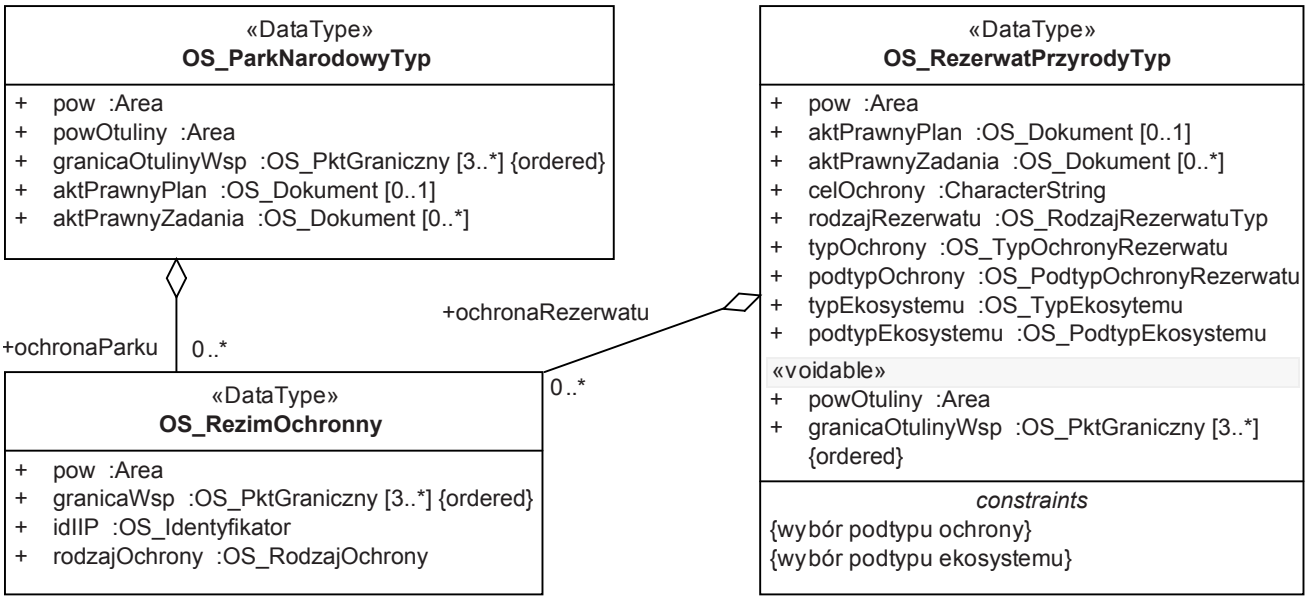
Podobna sytuacja występuje w przypadku klasy „AD_JednostkaAdministracyjna", czyli klas reprezentujących podział administracyjny kraju (województwa, powiaty, gminy). Różnica jest tylko taka, że w przypadku miejscowości przynależność jednej miejscowości do innej zdarza się sporadycznie, a w przypadku jednostek administracyjnych sytuacja taka występuje notorycznie (każda jednostka jest nadrzędna lub podrzędna, a najczęściej jest jednocześnie i nadrzędna, i podrzędna).

Zakładam, że Szacowny Czytelnik zgodzi się ze mną, iż w języku UML łatwiej jest przedstawić model, niż go opisywać. Wyjaśnienie znaczenia trzech „kresek" na rysunku zajęło prawie pół strony!

Relacje mogą zachodzić także pomiędzy klasami, które zostały utworzone tylko po to, by zdefiniować typ atrybutu. Z reguły takie klasy są zgromadzone w osobnym modelu i nie wchodzą w relacje z żadnymi innymi klasami. Na rys. 4.9 jest przedstawiona sytuacja ze schematu UML powstałego dla rejestru form ochrony przyrody. Klasy „OS_ParkNarodowyTyp" i „OS_RezerwatPrzyrodyTyp" definiują zakres informacyjny, jaki będzie gromadzony dla parków narodowych i rezerwatów przyrody (przedrostek OS, jak być może niektórzy Czytelnicy już odgadli, oznacza ochronę środowiska).

Jednocześnie jednak dla tych dwóch typów form ochrony mogą zostać zdefiniowane obszary podlegające ochronie ścisłej, czynnej lub krajobrazowej, które mają być opisywane tak samo. Został więc utworzony typ „OS_RezimOchronny", który jest częścią definicji typów dla parku narodowego i rezerwatu przyrody. W dodatku B (rys. B.2) umieściłem znacznie bardziej rozbudowaną strukturę relacji między zdefiniowanymi typami danych (która została zaczerpnięta ze specyfikacji danych opracowanych w ramach INSPIRE), a dotyczącą tematu z aneksu I do tej dyrektywy (SD – obszary chronione, 2010).

Dla relacji zachodzących pomiędzy klasami definiującymi typy atrybutów istnieje ograniczenie w ich stosowaniu. Relacje mogą zachodzić pomiędzy klasami o takich samych stereotypach (tak jest na rys. 4.9).

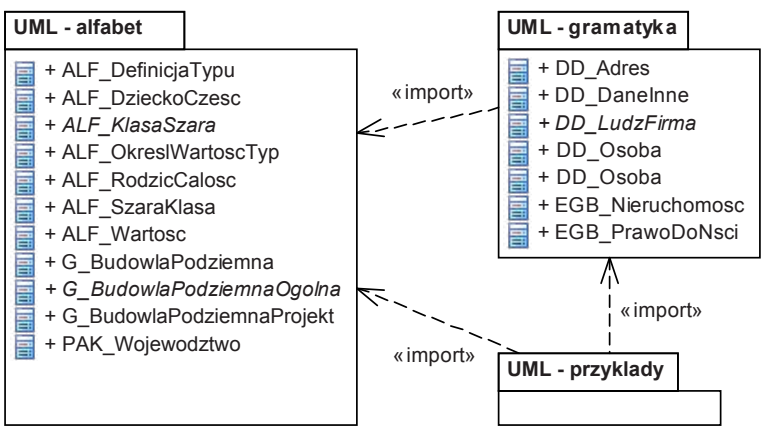


Rys. 4.9. Zależności pomiędzy typami danych (Formy ochrony, 2012)

Pakiety

Ostatnim elementem alfabetu UML, który chciałbym omówić, jest pakiet. Oznaczeniem pakietu jest prostokąt (przypomnijmy: klasa, pakiet – to prostokąty) z „małym" prostokątem przy górnej krawędzi, w którym znajduje się nazwa pakietu. Pakiet jest „pojemnikiem" zawierającym pewną liczbę klas, które stanowią całość z jakiegoś punktu widzenia. Stosuje się je przy tworzeniu złożonych modeli składających się z wielu klas. Zachodzi wtedy wręcz konieczność podziału takiego modelu na mniejsze części („podmodele"). Reprezentacją takich mniejszych modeli, które tworzą nasz poważny model (bo my przecież będziemy tworzyć tylko poważne modele), są pakiety. Wewnątrz prostokąta reprezentującego pakiet umieszcza się często nazwy klas wchodzące w skład pakietu (UML - alfabet i UML - Gramatyka z rys. 4.10).

Pakiety można łączyć relacjami (choć innymi niż te, które łączą klasy), by pokazać, jakie zależności zachodzą między nimi. Na rys. 4.10 są pokazane trzy pakiety, które utworzyłem. Klasy w nich służą do prezentacji ele-



Rys. 4.10. Pakiety wchodzące w skład modelu „Podręcznik"

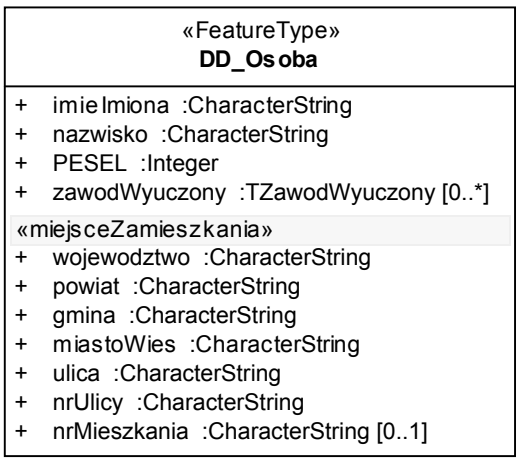
mentów języka UML i utworzonych schematów UML. Zostały one połączone relacjami oznaczonymi stereotypem «*import*», który oznacza, że klasy z pakietu „UML - gramatyka" będą „potrzebowały" klas z pakietu „UML - alfabet". Relacjom łączącym pakiety można nadać kilka różnych stereotypów – zostaną one omówione, gdy pojawią się na jakiejś stronie naszej książki. W dodatku A umieściłem diagram pakietów (rys. A.3.1) pokazujący odwoływanie się norm do specyfikacji technicznej 19103 oraz diagram (rys. A.3.2) ukazujący odwołania do normy PN-EN ISO 19107 dotyczącej geometrii i topologii.

Pakiety mogą zawierać inne pakiety (zależy to od złożoności modelu, który tworzymy).

4.2. Gramatyka, czyli jak „mówić" w UML

Jak mówić? Podobno najlepiej z sensem albo nic, ale to drugie rozwiązanie jest bardzo rzadko stosowane w praktyce. Wypadałoby zacząć od normy 19109 – Reguły schematów aplikacyjnych (ISO 19109, 2006), ponieważ w niej jest zapisana gramatyka języka UML. Innymi słowy, w normie 19109 jest zdefiniowana droga tworzenia schematów aplikacyjnych (co, gdzie i jak). Ale to jeszcze nie tutaj – za mało Państwo wiecie o UML, by się nie poddać, patrząc na *General Feature Model* z ISO 19109 (dodatek A, rys. A.4.1). Dlatego o tej normie będzie więcej w rozdziale o normach.

By coś powiedzieć, trzeba wiedzieć, co się chce powiedzieć, trzeba znać język (alfabet i przynajmniej trochę gramatyki) oraz choć parę słów – i można zacząć mówić. W UML wypowiedzi przyjmują postać schematu pojęciowego (zapisany model pojęciowy). Gdy już zapadnie decyzja, jakiego rodzaju obiekty z wybranego fragmentu świata rzeczywistego będą reprezentowane w modelu i jakie ich własności nas interesują, można przystąpić do konstruowania (budowania) schematu aplikacyjnego w UML.



Rys. 4.11. Klasa DD_Osoba (odsłona 1)

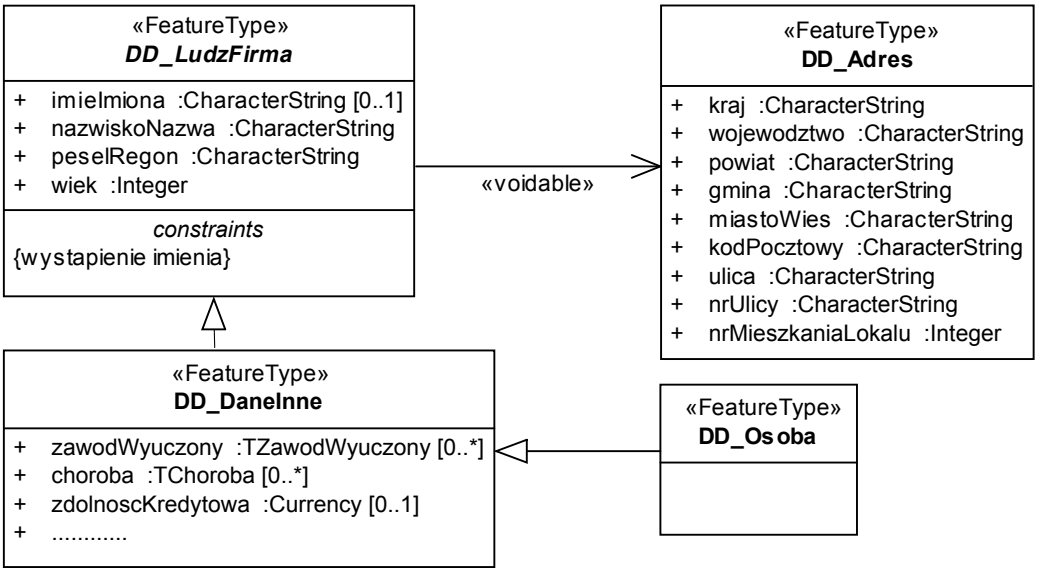
Załóżmy, że jednym z obiektów naszego schematu (rys. 4.11) będzie człowiek (w przypadku modeli, baz danych itp. wszystko to, co opisujemy atrybutami, jest klasą lub obiektem i nie zajmujemy się kwestiami podmiotów, osób „niezwykle ważnych" i „mniej ważnych"). Wszystko jest klasą (obiektem w bazie danych), jej atrybutem, relacją lub pakietem itp.

Przedrostek DD będzie reprezentował „dane demograficzne". Taki sposób opisywania klasy jak na rys. 4.11 jest bardzo mało elastyczny. Klasę DD_Osoba tworzą trzy odrębne części, które w sumie składają się na „osobę":

- dane „osobowe";
- dane „adresowe";
- dane „inne".

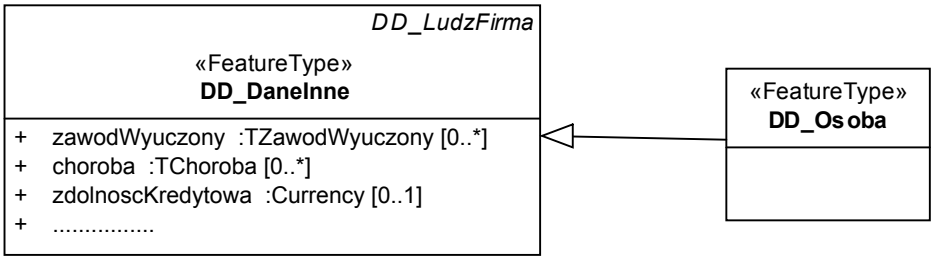
Jest to sytuacja, w której należy się zastanowić, czy może nie utworzyć dwóch lub trzech klas, by model stał się bardziej elastyczny i niektóre jego elementy mogły być wykorzystane w innych modelach.

Na rys. 4.12 przedstawiona jest ta sama sytuacja, ale w sposób bardziej elastyczny. Na pierwszy rzut oka może się to wydawać bardziej skomplikowane. Natomiast klasy „DD_Adres" i „*DD_LudzFirma*" (która jest klasą abstrakcyjną, co zapewne wszyscy zauważyli – nazwa klasy jest napisana *kursywą*), mogą zostać wykorzystane do definicji firm. Stąd pojawia się liczność [0..1] przy imieniu i ograniczenie, które „mówi", że imię ma liczność jeden, gdy informacje dotyczą osób. Chociaż na modelu klasa „DD_Osoba" nie ma żadnych atrybutów, to dzięki relacji dziedziczenia w bazie danych będzie opisywana atrybutami odziedziczonymi od klasy



Rys. 4.12. Klasa DD_Osoba (odsłona 2)

„*DD_LudzFirma*" i od klasy „DD_DaneInne". Jednocześnie, ponieważ „*DD_LudzFirma*" jest połączona relacją z klasą „DD_Adres", to klasa „DD_Osoba" też jest połączona relacją z klasą „DD_Adres" (dzięki dziedziczeniu od „ludzia"). Można, jeśli zajdzie taka potrzeba, sytuację przedstawioną na rys. 4.12 trochę „okroić" (rys. 4.13). Zamiast przedstawienia na tym modelu klasy „*DD_LudzFirma*" można pokazać w klasie „DD_DaneInne", że dziedziczy ona od klasy „*DD_LudzFirma*". Oczywiście klasa „*DD_LudzFirma*" w jakimś modelu musi zostać zdefiniowana.

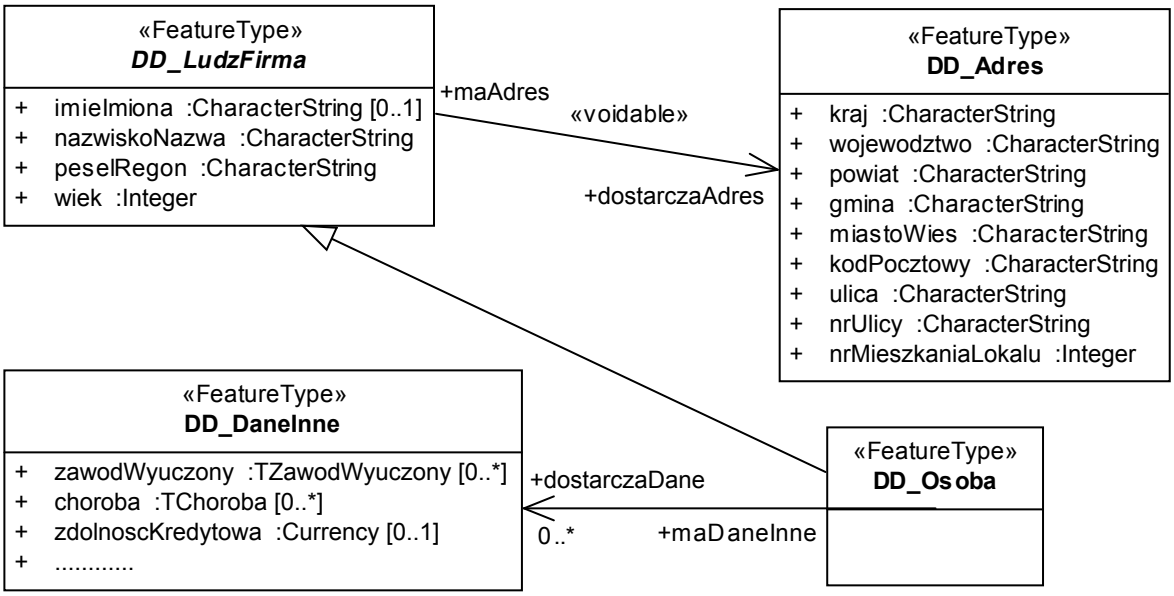


Rys. 4.13. Klasa DD_Osoba (odsłona 3)

Nie dla każdej osoby będziemy gromadzić dane inne, poza tym te dodatkowe dane mogą być różne, więc chyba lepiej jest dziedziczenie poprowadzić od osoby do klasy abstrakcyjnej, a do danych innych poprowadzić asocjację (dla części osób takie dane mogą być niepotrzebne). Po tych poprawkach model będzie wyglądał jak na rys. 4.14.

Na pewno najsłabszy (najmniej elastyczny) jest model (a w zasadzie klasa) z odsłony 1. Pozostałe modele są różne, ale żaden z nich nie jest skandalicznie zły. Na rys. 4.13, gdzie występuje dziedziczenie od klasy „DD_DaneInne", sytuację można „obronić" przed zarzutem, że nie dla wszystkich osób takie dane będą gromadzone, ponieważ liczności atrybutów w „DD_DaneInne" zaczynają się od [0..], więc mogą być osoby w bazie danych, dla których danych dodatkowych nie będzie. Ja uważam odsłonę 4 za najlepszą, ponieważ w sytuacji, gdy relacja pomiędzy dwiema klasami może w praktyce nie wystąpić, raczej nie należy w modelu umieszczać dziedziczenia. Asocjacja jest relacją jakby „łagodniejszą" niż relacja dziedziczenia, w której, jak powiedzieliśmy wcześniej, potomek dziedziczy wszystko. Takie określenie sugeruje, że potomek ma dziedziczyć, a nie, że raz będzie, a innym razem nie będzie dziedziczyć.

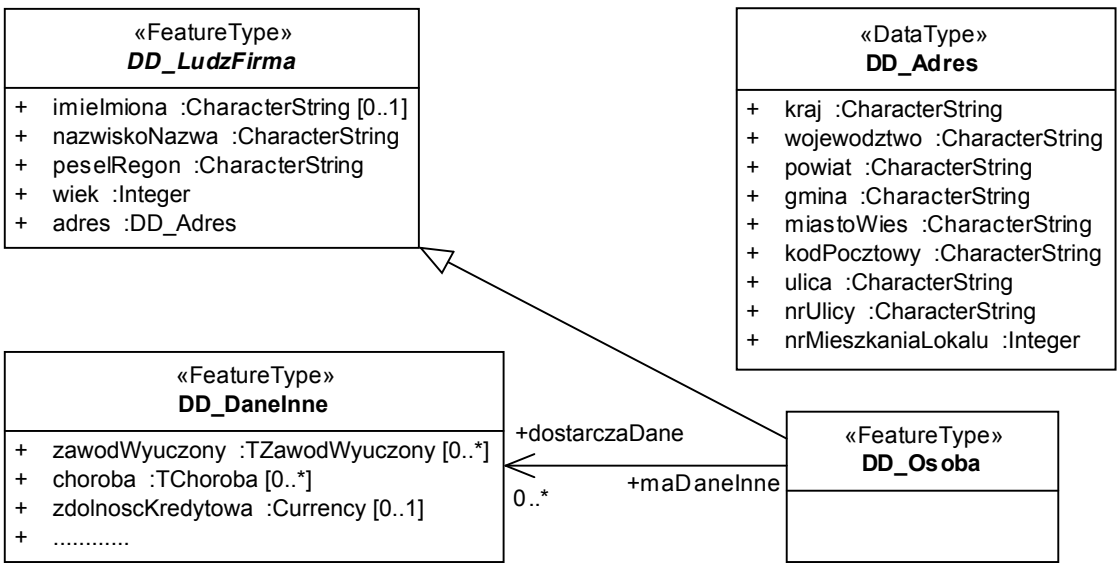
Kolejną sprawą jest odpowiedź na pytanie, co umieszczać na modelu: relację czy atrybut? Na rys. 4.14 występuje asocjacja pomiędzy „DD_Adres" a „*DD_LudzFirma*". Można to zamodelować inaczej – rys. 4.15. Zamiast relacji jest atrybut w „*DD_LudzFirma*" i zamiast klasy «*FeatureType*» jest klasa «*DataType*» definiująca typ atrybutu. Modele (raczej modeliki) z rys. 4.14 i 4.15 przedstawiają tę samą sytuację. Który model jest lepszy? Żaden! Można jedynie mówić tutaj o preferencjach. Ja wolę odsłonę 4, ponieważ, patrząc na model, wśród atrybutów nie muszę poszukiwać adresu. Relację widać od razu. Dla mnie sensowne jest zastępowanie relacji atrybutem, gdy klas i relacji jest dużo i każda dodatkowa zaciemniałaby model, a autor z jakiegoś powodu nie chce podzielić modelu na dwie części.



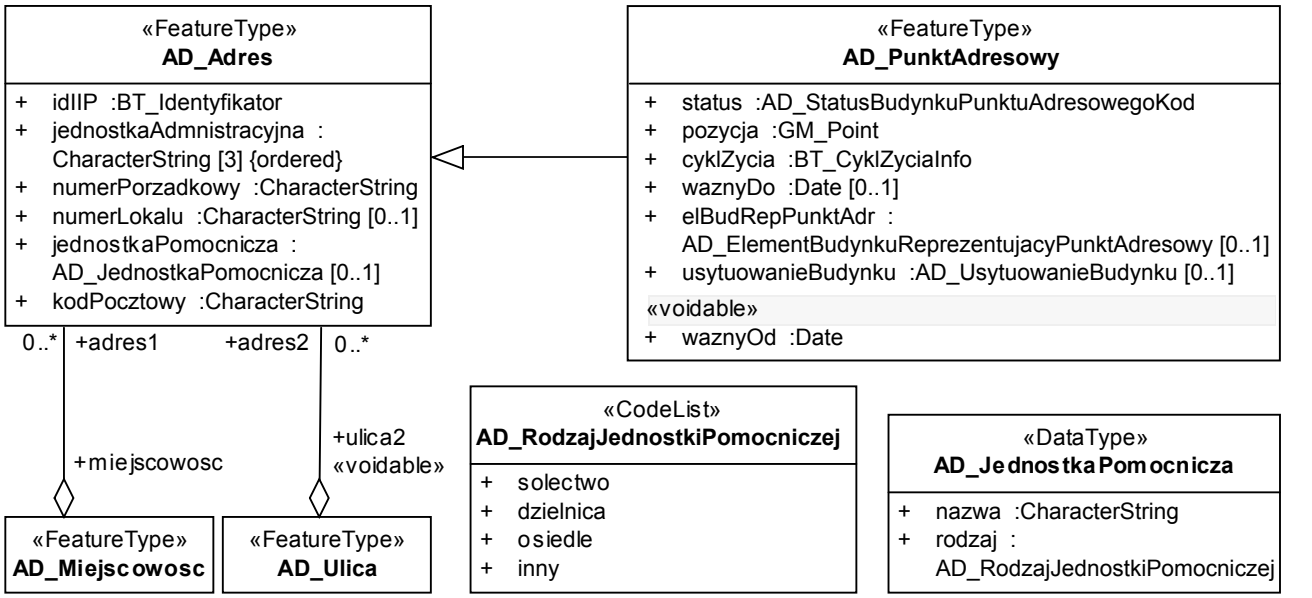
Rys. 4.14. Klasa DD_Osoba (odsłona 4)

Na koniec „znęcania" się nad osobą należy wspomnieć o harmonizacji między modelami. Tworząc schemat aplikacyjny, nie mogę udawać, że świat, który mnie interesuje, to jest tylko ten kawałek, który wybrałem do modelowania. **Tak nie wolno!** W tym pozostałym „kawałku" świata czasem dzieje się coś istotnego, np. zostało podpisane i opublikowane rozporządzenie ministra administracji i cyfryzacji z 9 stycznia 2012 r. w sprawie ewidencji miejscowości, ulic i adresów (EUMiA, 2012), a w nim jest schemat aplikacyjny UML. I jak należy się domyślać, będzie tam klasa definiująca adres – rys. 4.16. Jedynym sensownym działaniem jest wykorzystanie klasy „AD_Adres" w budowanym modelu, a nie tworzenie swoich własnych definicji, które może i będą oryginalne, ale w tym przypadku nie chodzi o oryginalność!

Na rys. 4.16 w klasie „AD_Adres" pojawia się bardzo ważny atrybut „idIIP", czyli **identyfikator infrastruktury informacji przestrzennej**. W zbiorach danych przestrzennych, których są miliony w państwach Unii Europejskiej, znajdują się opisy bilionów (jeśli nie więcej) obiektów przestrzennych. Aby była szansa na zapanowanie nad taką liczbą danych, każdy obiekt, który znajduje się w jakimkolwiek zbiorze danych przestrzennych, powinien posiadać unikalny identyfikator. Atrybut „idIIP" typu „BT_Identyfikator" jest takim atrybutem. Zapewnia on unikalność identyfikatora na terenie Unii Europejskiej, co oznacza, że każde drzewko, każdy krzaczek, jeśli jest odrębnym obiektem zapisanym w zbiorze danych przestrzennych, ma unikalny identyfikator. O konstrukcji typu „BT_Identyfikator", która zapewnia tę unikalność, podam więcej informacji w dalszej części książki.



Rys. 4.15. Klasa DD_Osoba (odsłona 5)

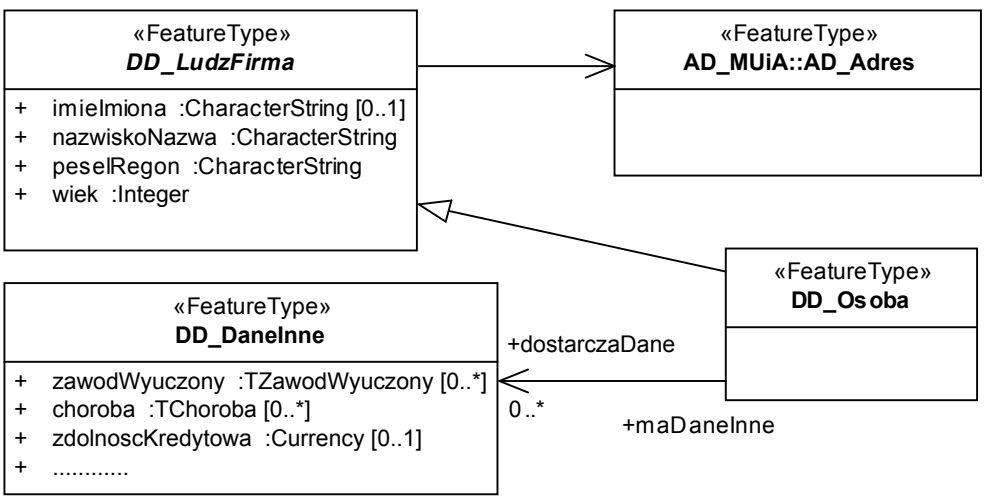


Rys. 4.16. Schemat aplikacyjny dla adresu (EMUiA, 2012)

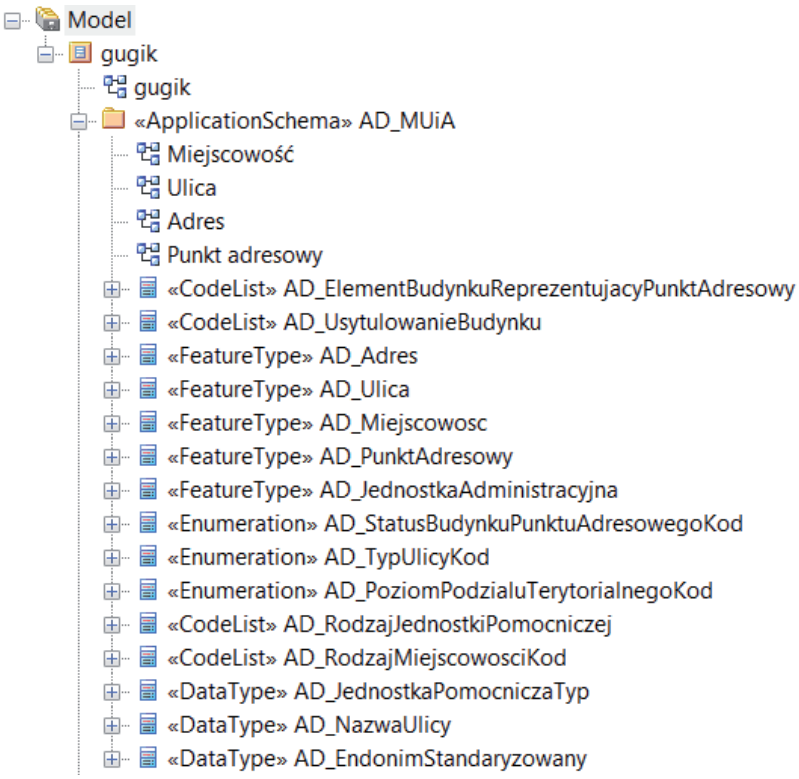
W ostatniej odsłonie klasa i jej otoczenie dotyczące osoby z uwzględnieniem rozporządzenia będą wyglądać jak na rys. 4.17. Przedstawiony na nim sposób odwołania się do innej klasy z innego rejestru nie jest sposobem jedynym. Są metody pozwalające odwoływać się do baz, dla których nie został jeszcze zbudowany schemat aplikacyjny (o tym też będzie nieco później).

Oprócz umiejętności opisywania obiektów (tzn. odpowiedniego doboru klas, które mają je reprezentować, i połączenia tych klas właściwymi relacjami) bardzo przydatna jest, a nawet powiedziałbym, że niezbędna, umiejętność podziału postawionego zadania na odpowiednią liczbę pakietów i modeli. Skoro powyżej skorzystaliśmy z przykładów z rozporządzenia EMUiA, zróbmy to ponownie. Na rys. 4.18 jest pokazany podział ewidencji miejscowości, ulic i adresów na modele. Rozporządzenie było opracowane w GUGiK (Główny Urząd Geodezji i Kartografii), stąd „główny" pakiet i model w nim mają nazwę „GUGiK". Pakiet „AD_MUiA" składa się z czterech modeli, które odpowiadają tematyce rozporządzenia. Można było połączyć dwa ostatnie modele, czyli „Adres" i „Punkt adresowy" w jeden, ale w sumie nic by to nie zmieniło. Model (czy – jak powinno być – schemat aplikacyjny) dla punktu adresowego jest przedstawiony na rys. 4.19 (schemat aplikacyjny dla adresu przedstawiono na rys. 4.16).

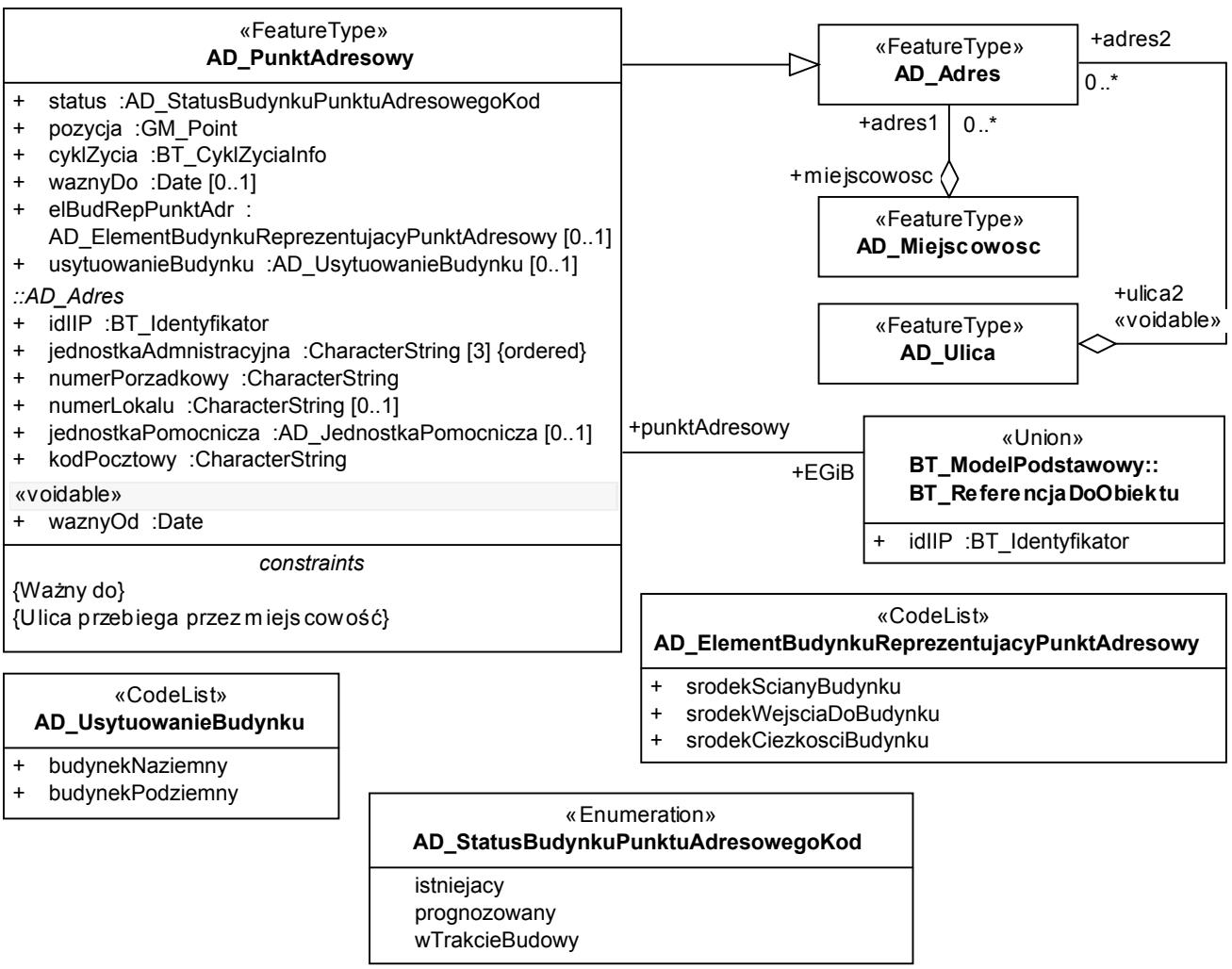
Niezbyt dobrym pomysłem jest tworzenie kilkunastu modeli, bo wtedy trzeba poświęcić trochę czasu (a może i więcej niż trochę) na uświadomienie sobie, jakie są modele i jakie klasy są w każdym z nich. Choć wyraźnie zaznaczyć trzeba, że nie można twierdzić, iż któryś z tych sposobów jest zły. Mnie duża liczba modeli przeszkadza, zwłaszcza gdy część z nich składa się np. z 1-3 klas. W przypadku EMUiA trzy czy cztery modele nie wpływają zauważalnie na poziom percepcji. Problemem staje się kilkanaście modeli.



Rys. 4.17. Klasa DD_Osoba (odsłona 6 – być może ostatnia)



Rys. 4.18. Pakiet miejscowości, ulic i adresów

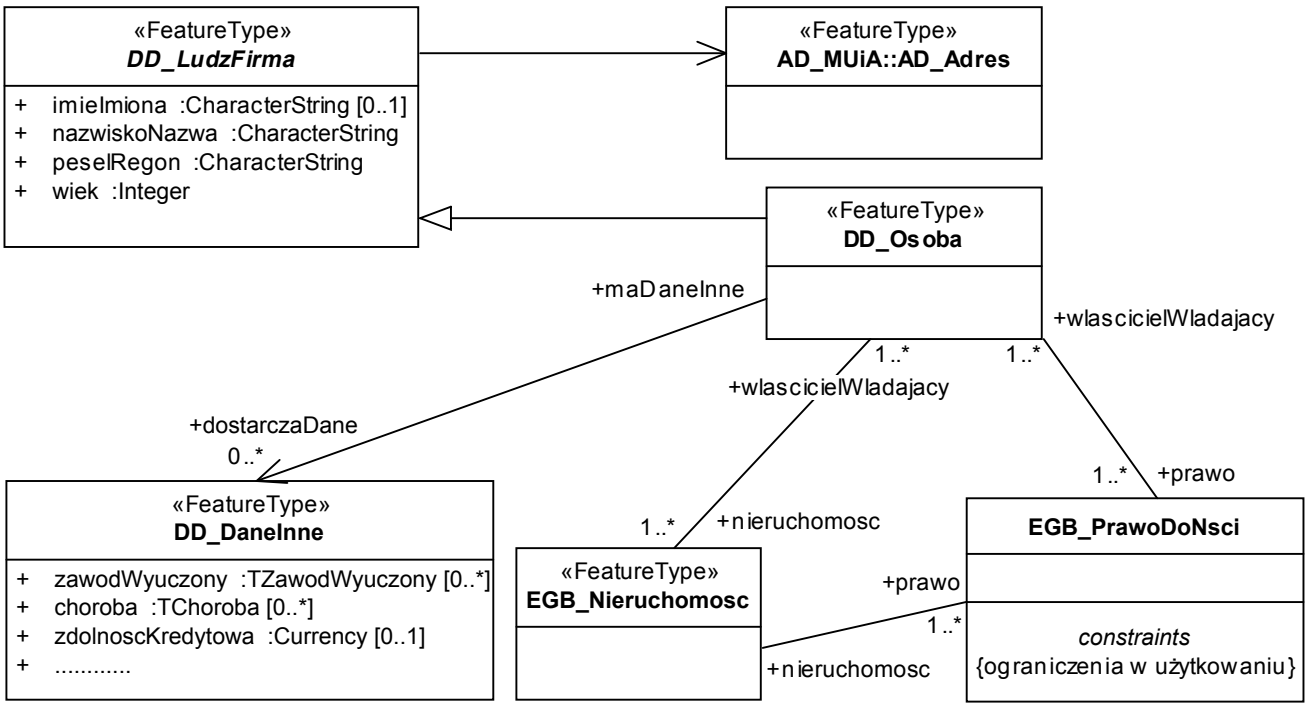


Rys. 4.19. Schemat dla punktu adresowego (EMUiA, 2012)

Na rys. 4.19 w klasie „AD_PunktAdresowy" są widoczne atrybuty odziedziczone z klasy adres. Źródłem powstania klasy punkt adresowy jest rozporządzenie (EMUiA, 2012), w którym rozróżniono dwa obiekty. Jeden z nich przechowuje dane adresowe (klasa „AD_Adres"), drugi głównie informacje „bazodanowe" oraz pozycję („AD_PunktAdresowy"). Do informacji „bazodanowych" zaliczam m.in.:

- „waznyDo" i „waznyOd" – są to daty dotyczące zmian prawnych, czyli np. dla atrybutu „waznyDo": „oficjalna data zakończenia prawnego obowiązywania funkcjonowania punktu adresowego" ;
- „cyklZycia" – oznacza „datę i czas, w których wersja obiektu została wprowadzona lub zmieniona w zbiorze danych".

Punkt adresowy – w przeciwieństwie do adresu – posiada położenie zdefiniowane za pomocą współrzędnych („pozycja: *GM_Point*" – klasa „*GM_Point*" jest zdefiniowana w normie 19107 i zostanie wyjaśniona w rozdziale dotyczącym norm ISO).



Rys. 4.20. Rozbudowa „naszego" modelu

Wracając do „naszego" modelu – jeśli go uzupełnimy o klasę reprezentującą nieruchomości, prawa do nieruchomości (co zostało schematycznie pokazane na rys. 4.20), to można powiedzieć, że powstaje zalążek schematu aplikacyjnego dla ewidencji gruntów i budynków.

W tym miejscu można i należy wspomnieć o strategiach modelowania. Pod pojęciem „strategia" rozumiem wybór podstawy (sposobu) modelowania. Co to oznacza? Ścierają się u nas zwolennicy dwóch różnych strategii:

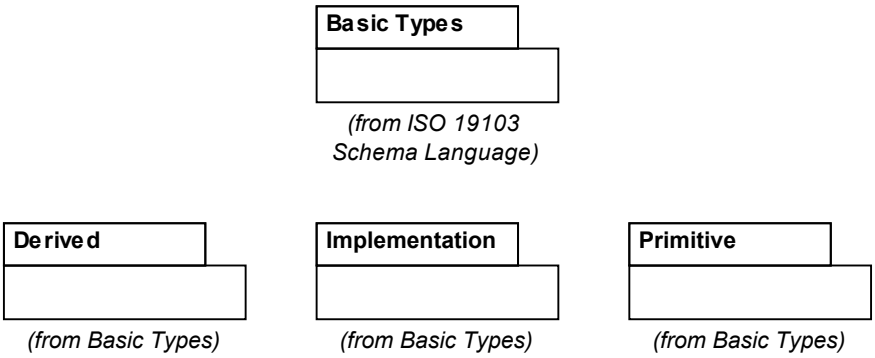
- Obierzmy jako podstawę naszych, polskich modeli specyfikacje danych (o których będzie w rozdziale o INSPIRE). Specyfikacje są modelami opisującymi obiekty, które mają się znaleźć w zbiorach zgodnych z INSPIRE i które to zbiory będą opublikowane w ramach INSPIRE. Specyfikacje obejmują 34 tematy danych zdefiniowane w aneksach do dyrektywy INSPIRE (Dyrektywa INSPIRE, 2007). Dołóżmy do tych specyfikacji to, czego tam z naszego punktu widzenia brak, i mamy schematy aplikacyjne.
- Zbierzmy specjalistów z danej dziedziny (którzy nie muszą się znać na UML i specyfikacjach danych), poinformujmy ich, że chodzi nam o uspójnienie zbiorów (dane są zbierane raz i zapisywane w jednym rejestrze). Specjaliści opiszą obiekty i ich właściwości, my na tej podstawie utworzymy schematy aplikacyjne UML.

Obie strategie mają swoje zalety i wady. Generalnie w GUGiK, choć nie tylko tam, została wybrana i jest realizowana strategia druga. Niewątpliwie jednak bardzo dobrym pomysłem jest starać się „zerkać" w stronę INSPIRE w trakcie tworzenia schematów aplikacyjnych UML.

Wracając do ewidencji gruntów i budynków – jest opracowana, przyjęta przez odpowiednie gremium i opublikowana specyfikacja danych dotycząca działek katastralnych (tak są w Unii nazywane działki ewidencyjne) oraz jest przygotowywana norma ISO 19152 dotycząca katastru. Już w 2006 r. pojawiła się w magazynie „GIM" informacja na ten temat z propozycją modelu UML (Lemmen Ch. i. in., 2006) – zob. dodatek B (rys. B.1). Pytanie: tworzyć własny schemat czy oprzeć się na działkach katastralnych i projekcie normy?

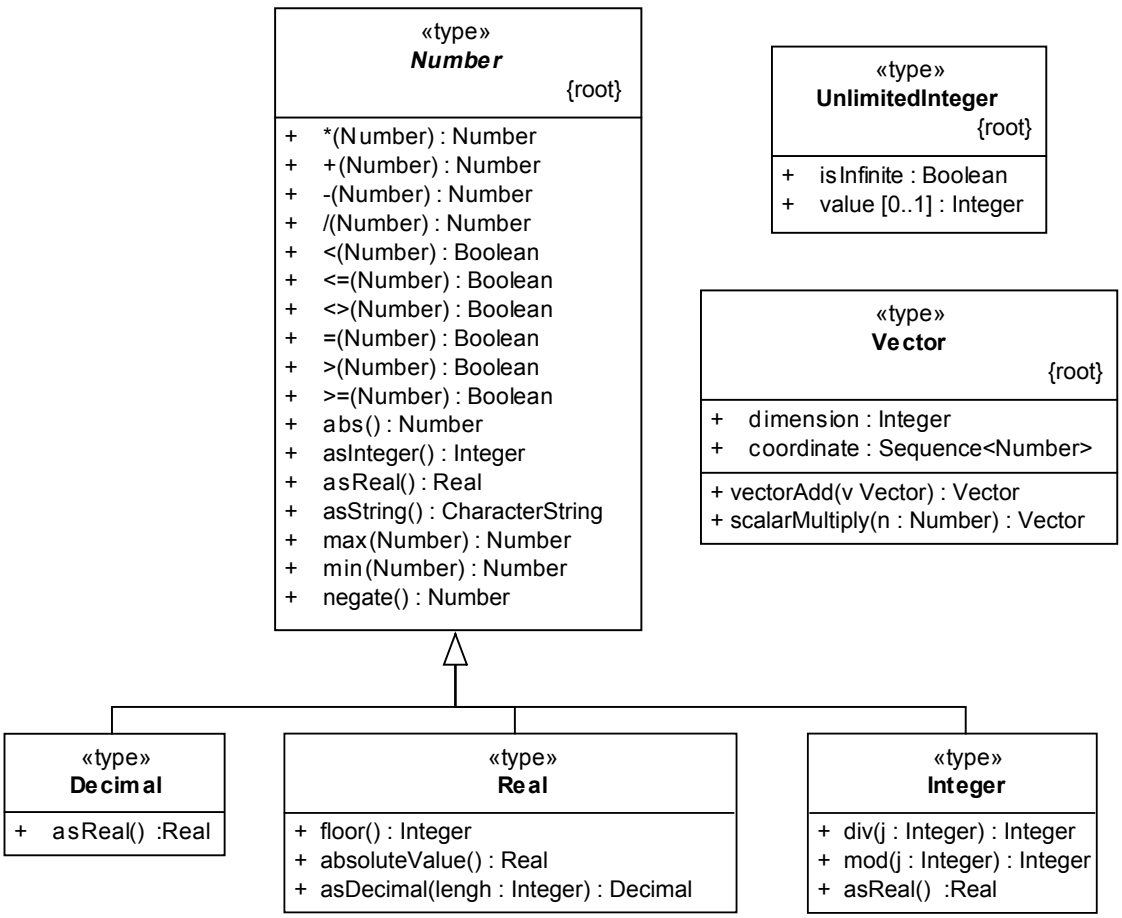
4.3. Specyfikacja techniczna ISO/TS 19103

W specyfikacji ISO/TS 19103 zostały zdefiniowane cztery pakiety (rys. 4.21). O elementach (klasach) z pakietów *Basic Types* i *Primitive* zostało nieco powiedziane już wcześniej, do *Primitive* jeszcze wrócimy. Zostały nam więc do omówienia jeszcze dwa pakiety.

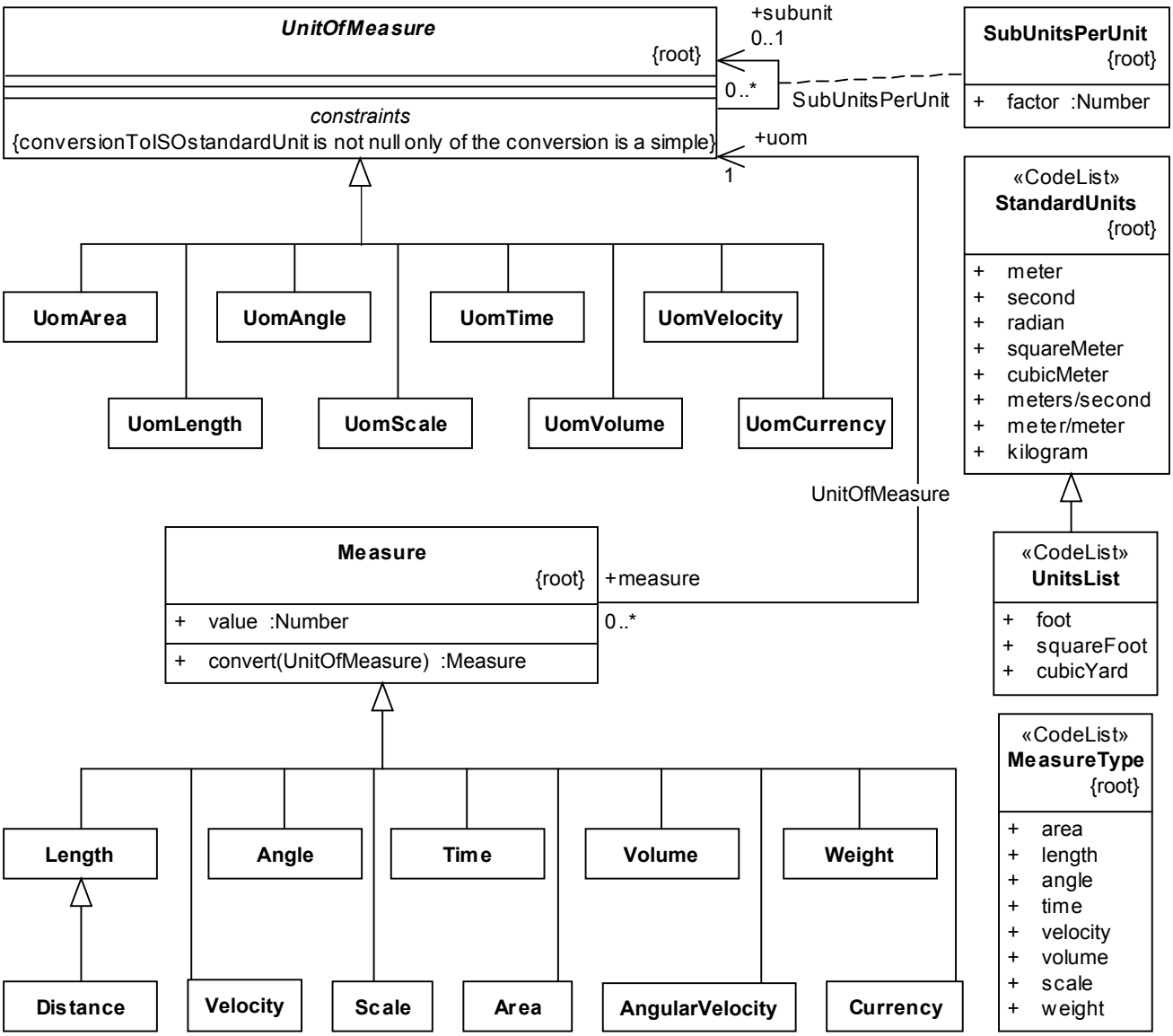


Rys. 4.21. Pakiety ze specyfikacji technicznej 19103

Większość klas w specyfikacji ma stereotyp «*Type*», który w zasadzie nie jest stosowany w budowanych schematach aplikacyjnych. Jest on nadawany klasom, które są wykorzystywane do zdefiniowania dziedziny jakiegoś elementu (np. atrybut może mieć typ *Integer*, czyli jego dziedziną są liczby całkowite). Klasy te mogą mieć abstrakcyjne atrybuty i relacje, co oznacza, że potem muszą one zostać skonkretyzowane, np. atrybut może być typu *Number*, ale klasa definiująca ten typ jest klasą abstrakcyjną, więc nie może zostać zaimplementowana. Z kolei w schemacie nie może wystąpić obiekt, np. *Real*, taki może być typ obiektu albo raczej typ atrybutu. Zaimplementowane mogą zostać podtypy *Number* (czyli któraś z jej klas pochodnych – zob. rys. 4.22).



Rys. 4.22. Pakiet *Numeric* (ISO/TS 19103, 2005)



Rys. 4.23. Pakiet *Derived*

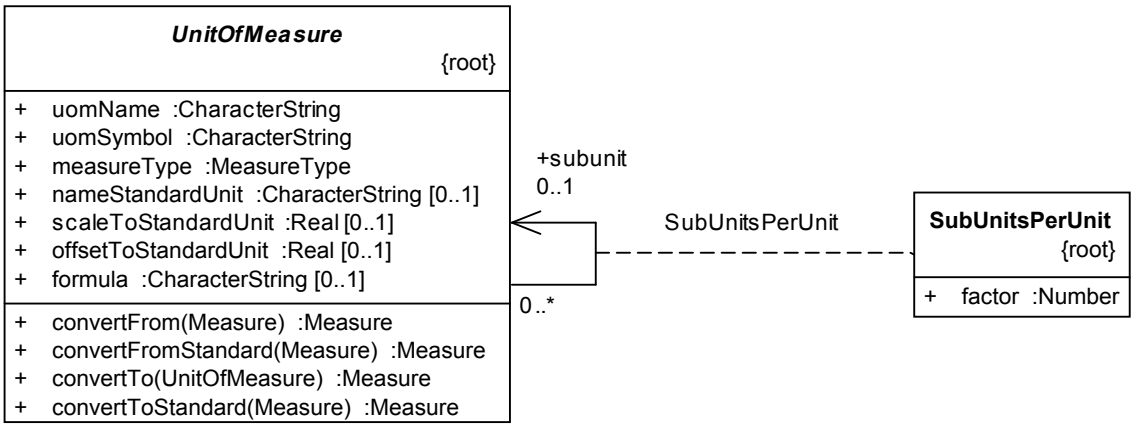
W klasie abstrakcyjnej *Number* są zdefiniowane operacje (rys. 4.22), które w systemie bazodanowym powinny zostać zaimplementowane, by w każdym przypadku, gdy dane będą w postaci liczb, można było te operacje wykonać. Czyli każdą operację na każdym typie „liczbowym": dziesiętnym (*Decimal*), całkowitym (*Integer*) i rzeczywistym (*Real*). Oczywiście należy tu uwzględnić specyfikę typów – trudno np. oczekiwać wykonalności dzielenia dla typu całkowitego.

Pakiety dla *Enumeration* i dla typów logicznych (*Truth*) są przedstawione w dodatku A w punkcie 2.

W pakiecie *Derived* zostały zdefiniowane klasy obejmujące wyniki różnych pomiarów i jednostki wykorzystywane przy zapisie tych wyników (rys. 4.23). Występuje tam klasa *Measure*, od której dziedziczą wszystkie klasy opisujące różnego rodzaju wyniki pomiarów: powierzchni (*Area*), kąta (*Angle*) czy długości (*Length*). Ogólnie *Length* jest długością elementów krzywoliniowych, a szczególnym jej przypadkiem jest długość (lub czasem odległość) elementów prostoliniowych (*Distance*). Są też wyniki pomiaru skali (*Scale*), prędkości (*Velocity*).

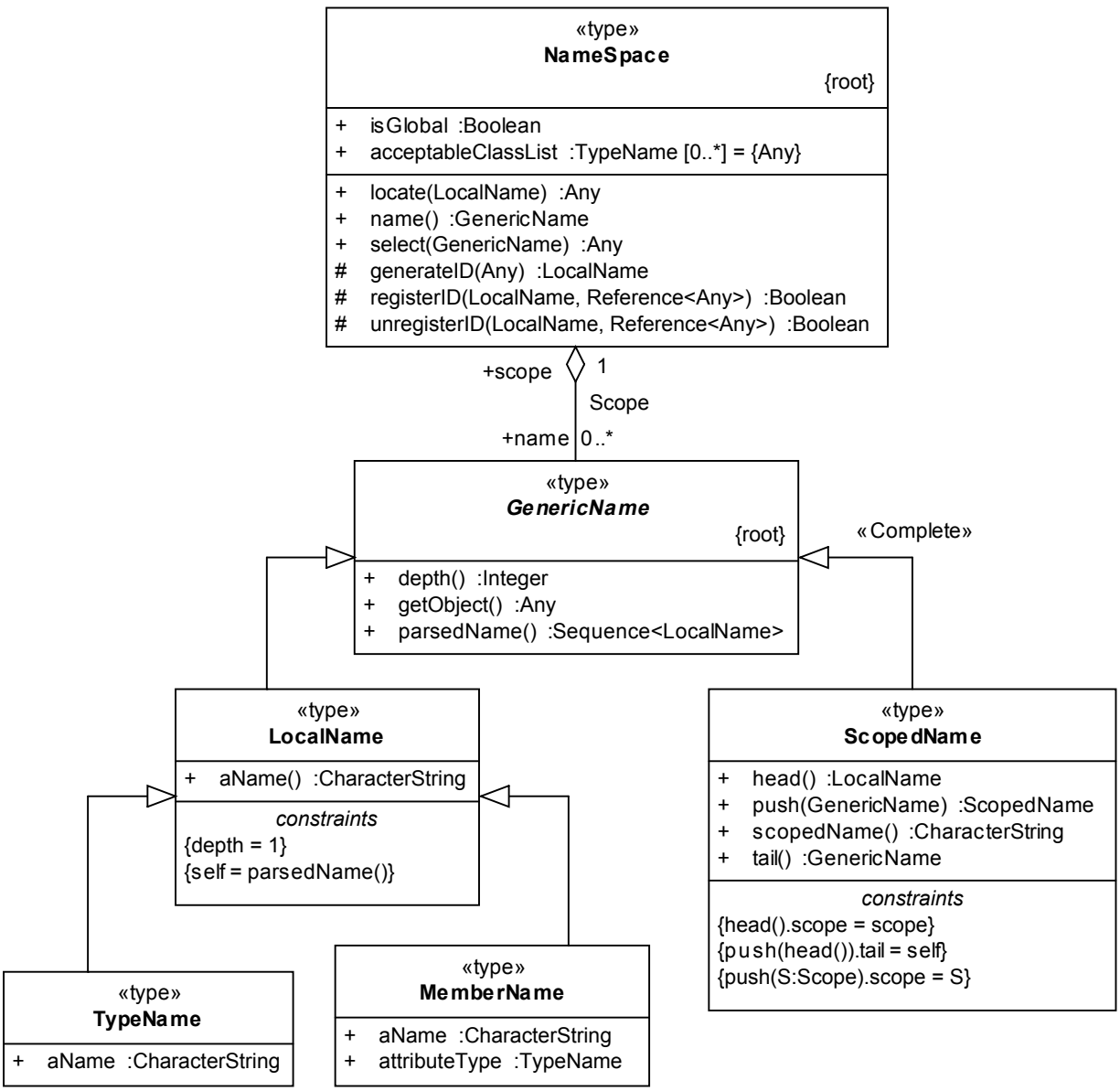
Klasa *Measure* jest połączona relacją z *UnitOfMeasure*, która z klasami pochodnymi definiuje różne jednostki (miana), w jakich są podawane wartości. To oznacza bardzo wygodną sytuację dla tworzących model. Wystarczy jako wynik pomiaru długości podać *Distance*, a nie *Real* i nie trzeba potem się martwić i słuchać uwag różnych „fachowców", że się zapomniało o jednostkach. Pełna definicja *UnitOfMeasure* jest pokazana na rys. 4.24

Kolejnym pakietem jest *Implementation*, w którym znalazły się podpakiety obejmujące typy zbiorowe *Collection*, nazwy *Names* oraz typy rekordowe (typ znany m.in. z różnych języków programowania) *Records and Class Metadata*. Poniżej opiszę pokrótce pakiet nazw, by pokazać, że nawet tworzenie i nadawanie nazw zostało zestandaryzowane, by jak najmniej zostawić przypadkowi czy inwencji osób konstruujących schematy aplikacyjne.



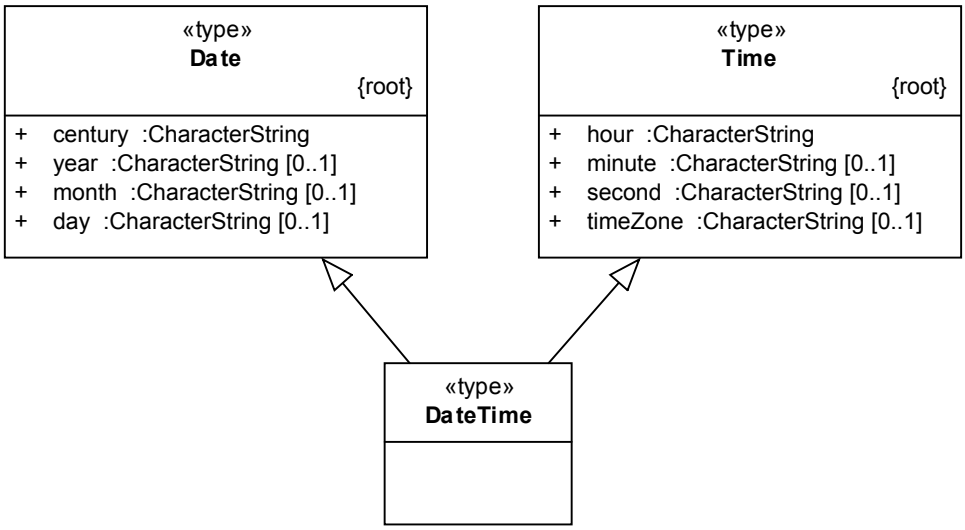
Rys. 4.24. Klasa *UnitOfMeasure*

Przestrzeń nazw tworzy strukturę nazw typów i atrybutów, w której „nazwy" są ciągami (łańcuchami) znaków i mogą być dodawane do obiektów poprzez operację *getObject*. Schemat dla przestrzeni nazw, nazw ogólnych (*GenericName*) i lokalnych (*LocalName*) jest pokazany na rys. 4.25. Klasa *GenericName* i jej „dzieci" mają tylko operacje i ewentualnie ograniczenia.



Rys. 4.25. Schemat przestrzeni nazw

GenericName jest rodzicem (klasą abstrakcyjną) dla klas *LocalName* i *ScopedName*. Ta druga jest kompozytem służącym do zlokalizowania innej przestrzeni nazw, do której występuje odwołanie z danej przestrzeni. *LocalName* odwołuje się do lokalnych obiektów dostępnych w danej przestrzeni nazw. Z kolei *TypeName* odnosi się do typów obiektów wykorzystywanych w danym schemacie aplikacyjnym.



Rys. 4.26. Klasy opisujące czas

W pakiecie *Basic Types* są zdefiniowane klasy opisujące położenie w czasie. Czas jest niezwykle ważnym elementem charakteryzującym informacje przestrzenne. Na tyle podstawowym, że część klas pozwalających opisać atrybuty dotyczące czasu zdefiniowano w specyfikacji 19103. Są to trzy klasy: *Date*, *Time* i *DateTime* (pomijam tutaj inne kwestie, np. związane z precyzją określenia czasu), których nazwy bezpośrednio identyfikują, czego klasy dotyczą (rys. 4.26). Większość atrybutów ma liczność [0..1] i ich występowanie w bazie danych zależy od precyzji, z jaką czas znamy. Zarówno data, jak i czas są łańcuchami znaków (typ *CharacterString*) zgodnymi z normą ISO 8601. Datę zapisujemy jako: 2012-11-24, a czas 12:12:12 lub 12:12:12+01:00, gdy podajemy czas przesunięty o godzinę w stosunku do zdefiniowanego czasowego układu odniesienia. Datę i czas podajemy w formacie: 2012-11-24T12:12:12.

Bez praktycznego wdrożenia, czyli zbudowania modelu i zapisania go w UML-u, nie ma potrzeby dalszego teoretyzowania. Wydaje mi się (raczej mam nadzieję), że materiał z tego rozdziału będzie Państwu pomocny w nauce czytania modeli UML i „mówienia" w UML.

Tym, którzy czują niedosyt (i ja im się nie dziwię), mogę zaproponować stronę internetową na temat UML http://www.uml.org/, gdzie opublikowano np. dwa dokumenty:

- „UMLInfrastructure – UML2.4–2010–11–16.pdf",
- „Superstructure – UML2.4 – 2010–11–14.pdf".

W każdym z nich znajdziecie Państwo dużo informacji o UML i jego składowych.

Zobowiązałem się wytłumaczyć, jak należy rozumieć stereotyp «*Union*», gdzie go należy stosować, i wyjaśnień brak, co zapewne zauważyli prawie wszyscy Czytelnicy. Niestety, tak się zdarzyło. Stereotyp «*Union*» zostanie wyjaśniony w rozdziale o normach w przypadku klasy *GM_Position*. Zapewne wiecie, jak to jest: kierownik Parzyński, a podwładny Zenek i jak mu się zechce, to zrobi, a jak nie, to nie, a później trzeba się tłumaczyć.