Tomasz GRATKOWSKI
UNIWERSYTET ZIELONOGÓRSKI, INSTYTUT INFORMATYKI I ELEKTRONIKI

Wykorzystanie modeli ontologicznych w procesie uzupełniania zbioru wiedzy niezbędnego w procesie projektowania systemu informatycznego

Dr inż. Tomasz GRATKOWSKI

Uzyskał stopień doktora nauk technicznych w zakresie informatyka na Wydziale Informatyki i Zarządzania Politechniki Wrocławskiej. Autor jest adiunktem w Instytucie Informatyki i Elektroniki Uniwersytetu Zielonogórskiego oraz współpracuje z firmami z branży informatycznej. Głównymi tematami zainteresowań naukowych są: projektowanie systemów informatycznych, metodyki projektowania używane w inżynierii oprogramowania oraz systemy rozproszone.

e-mail: T.Gratkowski@iie.uz.zgora.pl

Streszczenie

W ramach artykułu została zaprezentowana koncepcja zastosowania zbiorów modeli ontologicznych, które pozwalają na uzupełnienie zbioru informacji niezbędnego w procesie projektowania. W przypadku zastosowania klasycznego sposobu projektowania, zbiór ten jest uzupełniany na podstawie wiedzy projektanta. Zbiór informacji stanowi niezbędne uzupełnienie przetwarzanego modelu wiedzy przez system automatyzujący proces projektowania systemu komponentowego.

Słowa kluczowe: metodyki projektowania systemów informatycznych, systemy wspomagające proces projektowania, modelowanie.

Ontology models for a supplement set of knowledge required for design of software

Abstract

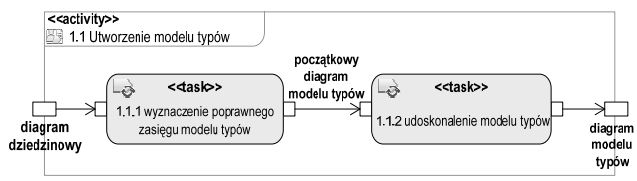
The process of design of software is a part of the software development methodology. It is divided into tasks by means of Chessman-Daniels [1] (CD) methodology. Each task has a set of input and output artifacts (both are UML [8] models). The paper presents a part of research dealing with issues of performing tasks automatically from CD methodology. The main idea to automatise tasks uses rules [2, 3]. The rules define how the task should be executed, and how to use information stored in input artifacts to create output artifacts. The research shows that some tasks require additional information to create output artifacts. In a classical process of design when software is designed by man, the designer's experience and knowledge create additional information. When the design process is performed automatically, an additional set of information broadening the system's knowledge about the designed software is needed. In the paper there is also presented the analysis of using ontologies [4-7] to describe the set of additional information. The ontology is a formal representation of a set of concepts within a domain and the relationships between those concepts. The ontologies used are described in OWL [7] (Fig. 6). There is given an algorithm (Figs. 4, 5) for using knowledge from ontologies to make automatically the "complete of multiplicity for association" step from "improve of types model" task (Fig. 1). This step is the smallest part of a software development flow [2], and it is a part of the task. The algorithm finds multiplicities for an association in the set of ontologies. The requirements for the input and output artifacts are given in Figs. 2, 3 in OCL [9] language.

Keywords: software design methodology, computer-based support in the software design process, software modeling.

1. Wstęp

Zastosowanie metodyk projektowania usystematyzowało i uprościło proces projektowania. Pomimo używania narzędzi i technologii wspomagających, proces projektowania oparty jest nadal głównie na wiedzy i doświadczeniu projektantów. Prowadzone przez autora badania, zmierzają do zautomatyzowania wybranych fragmentów procesu projektowania tak, aby proces ten mógł odbywać się bez udziału człowieka. W tym celu dla wybranej metodyki Cheesmana-Danielsa (CD) [1] dokonano niezbędnej formalizacji struktury metodyki [2] oraz przetwarzanych w ramach metodyki artefaktów [3]. Zadania realizowane w metodyce zostały opisane przy użyciu reguł [3]. Proces projektowania, podzielony na zadania, definiuje artefakty wejściowe i wyjściowe dla każdego z zadań. Artefakty wejściowe zawierają zbiór informacji na podstawie, których zbiór reguł przypisany do zadania wykonuje czynności, które umożliwiają wygenerowanie artefaktu wyjściowego. W wyniku dokonanej analizy zadań dla pierwszego etapu z metodyki Cheesmana-Danielsa, wyodrębniono zadania w ramach, których zbiór informacji z artefaktów wejściowych był niekompletny dla realizowania zadania w sposób automatyczny. Przykładowym zadaniem było zadanie „udoskonalanie modelu typów", które składa się z dwóch kroków „uzupełnianie typów dla atrybutów" oraz „uzupełnienie liczebności dla asocjacji". Krok jest najmniejszym niepodzielnym elementem definiującym prze-pływ czynności w ramach przyjętej formalizacji metodyki [2].

W przypadku obu kroków wystąpił problem braku niezbędnych informacji w artefaktach wejściowych tak, aby zadanie mogło zostać wykonane automatycznie. Na rysunku 1 przedstawiono zadanie 1.1.2 „udoskonalanie modelu typów". Głównym celem zadania jest uzupełnienie modelu danych o niezbędne dla dalszego procesu projektowania informacje. W przypadku, gdy proces projektowania realizowany jest w klasyczny sposób, projektant wykonuje zdanie na podstawie posiadanej wiedzy i doświadczenia. W przypadku zastosowania systemu wspomagającego, automatyzującego wykonanie zadania, należało opracować mechanizm uzupełnienia zbioru wiedzy o niezbędne informacje. W poniższym artykule zaprezentowano rozwiązanie problemu braku niezbędnych informacji dla kroku „uzupełnienie liczebności dla asocjacji".



Rys. 1. Czynność 1.1 – utworzenie modelu typów
Fig. 1. Activity 1.1 – creating a type model

Artefaktem wejściowym dla zadania 1.1.2 jest początkowy diagram modelu typów [1] (PDMT), który będąc diagramem klas [8], reprezentuje pojęcia, definiujące zasięg projektowanego systemu (ograniczenia w języku OCL [9] dla diagramu zostały przedstawione na rysunku 2). Po wykonaniu zadania 1.1.2 na wyjściu uzyskiwany jest diagram modelu typów (DMT) [1], który również jest diagramem klas. Zgodnie z przyjętymi ograniczeniami (rys. 3), w odróżnieniu od początkowego diagramu modelu typów, DMT musi spełniać poniższe wymagania:

- DMT3 – każdy atrybut musi posiadać nazwę oraz musi posiadać przypisany typ,
- DMT5 – asocjacje muszą posiadać określoną liczebność na obu końcach oraz muszą posiadać niepowtarzalne nazwy,
- DMT7 – dozwolone elementy na diagramie: to klasy, typy podstawowe, asocjacje i dołączone profile.

Krok „uzupełnienie liczebności dla asocjacji" jest odpowiedzialny za uzupełnienie artefaktu wejściowego tak, aby spełniał wymaganie DMT5.

```
-- PDMT1 Diagram klas zawiera tylko klasy oraz
przedstawia powiązania (asocjacje) między nimi.
context Classifier
 inv PDMT1_1: self.oclIsKindOf(Class) or
   self.oclIsKindOf(Relationship)
context Relationship
 inv PDMT1_2: self.oclIsTypeOf(Association) or
   self.oclIsTypeOf(ProfileApplication)
context Association
 inv PDMT1_3: self.memberEnd->size()=2
-- PDMT2 Klasy muszą posiadać niepowtarzalną nazwę.
context Class
 inv PDMT2_1: self.name.size()>0
context Package
 inv PDMT2_2: Class.allInstances()->forAll(c1, c2 | c1
   <> c2 implies c1.name <> c2.name)
-- PDMT3 Atrybuty muszą posiadać niepowtarzalną nazwę.
context Property
 inv PDMT3_1: self.owner.oclIsKindOf(Class) implies
   self.name->size() > 0
context Class
 inv PDMT3_2: self.attribute->forAll(c1, c2 | c1 <> c2
   implies c1.name <> c2.name)
-- PDMT4 Asocjacje mogą posiadać określoną liczebność
oraz nazwy.
-- PDMT5 Klasy muszą posiadać stereotyp 'type' i nie
mogą zawierać operacji.
 inv PDMT5_1:self.isStereotypeApplied(
   getApplicableStereotype('CDMProfile::type'))
 inv PDMT5_2: self.getOperations()->size() = 0
```

Rys. 2. Ograniczenia dla początkowego diagramu modelu typów
Fig. 2. Constraints for the initial diagram of the type model

```
--DMT1 Wymagania i ograniczenia takie jak PDMT1, PDMT2,
PDMT3, PDMT4
--DMT2 Klasy muszą posiadać stereotyp «type».
context Class
 inv DMT2_1: self.isStereotypeApplied
   (getApplicableStereotype('CDMProfile::type'))
--DMT3 Kazdy atrybut musi posiadać nazwę oraz musi
posiadać przypisany typ.
context Class
 inv DMT3_1: self.attribute->notEmpty() implies self.
   attribute->forAll(atr:Property| atr.name.size()>0
    and atr.type->notEmpty())
--DMT4 Użyte atrybuty mogą być atrybutami
sparametryzowanymi.
context Class
 inv DMT4_1: self.getOperations()->size() > 0 implies
   self.getOperations()->forAll(isStereotypeApplied
    (getApplicableStereotype('CDMProfile::att')))
--DMT5 Asocjacje muszą posiadać określona liczebność na
obu końcach oraz muszą posiadać niepowtarzalne nazwy.
context Association
 inv DMT5_1: self.allFeatures().oclAsType(Property).
   upperValue->forAll(vs:ValueSpecification|vs<>null)
 inv DMT5_2: self.allFeatures().oclAsType(Property).
   lowerValue->forAll(vs:ValueSpecification|vs<>null)
 inv DMT5_3: self.name.size()>0
context Package
 inv DMT5_4: Association.allInstances()->forAll(c1, c2
   | c1 <> c2 implies c1.name <> c2.name)
--DMT6 Asocjacje mogą posiadać przypisane ograniczenia.
--DMT7 Dozwolone elementy to klasy, typy podstawowe,
asocjacje i dołączone profile.
context Classifier
 inv DMT7_1: self.oclIsKindOf(Class) or
   self.oclIsKindOf(Relationship) or
    self.oclIsKindOf(PrimitiveType)
context Relationship
 inv DMT7_2: self.oclIsTypeOf(Association) or
   self.oclIsTypeOf(ProfileApplication)
context Association
 inv DMT7_3: self.memberEnd->size()=2
```

Rys. 3. Ograniczenia dla diagramu modelu typów
Fig. 3. Constraints for the diagram of the type model

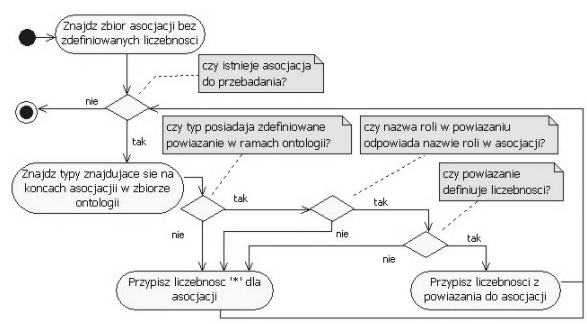
Bardzo często na poziomie definicji wymagań projektowanego systemu, które są opisane przy użyciu diagramu dziedzinowego (na podstawie, którego tworzony jest początkowy diagram dziedzinowy), analityk systemowy zdobywający wymagania nie umieszcza wszystkich niezbędnych informacji potrzebnych na późniejszych etapach projektowania. Wynika to z powodu braku możliwości zdefiniowania tych informacji na podstawie analizy potrzeb klienta lub z powodu pozostawienia decyzji o sposobie zaprojektowania systemu, które podjęte zostaną na późniejszych etapach projektowania. W przypadku nieumieszczenia liczebności dla każdego końca asocjacji w procesie zdobywania wymagań, brakujące liczebności muszą być uzupełnione w kroku „uzupełnienie liczebności dla asocjacji".

Termin ontologie pojawił się w kontekście informatyki w badaniach dotyczących modelowania danych [4]. Ontologia zajmuje się odkrywaniem i opisywaniem wybranych fragmentów rzeczywistości. Opisy mogą być realizowane na różnych poziomach szczegółowości. Istnieje wiele typów ontologii oraz wiele sposobów ich tworzenia. Ontologia musi być wyrażona przy użyciu zdefiniowanego formalnego języka ontologii. Główne komponenty stosowane w ramach ontologii: jednostka - instancja, klasa, atrybuty - własności, związki, ograniczenia, reguły, aksjomaty, zdarzenia. Przykładami języków ontologii nawiązującymi do języka logiki są Knowledge Interchange Format [5] oraz Common Logic [6], natomiast przykładem języka nawiązującego do sieci semantycznych jest Web Ontology Language (OWL) [7].

Zbiory ontologiczne zgodnie ze swoim głównym przeznaczeniem służą do opisywania wybranych fragmentów rzeczywistości. Uzyskany w procesie projektowania model systemu zapisany przy użyciu języka UML [8] jest również przykładem języka specyfikacji ontologii. Dlatego w przypadku braku niezbędnych informacji w modelach UML, informacje mogą zostać odszukane w innych zbiorach ontologicznych. Ze względu na dużą popularność i dostępność oraz ze względu na możliwość definiowania cech relacji pomiędzy definiowanymi pojęciami zdecydowano się na wykorzystanie modeli ontologicznych opisanych w języku OWL. Zbiory ontologiczne opisujące tą samą domenę problemu, jakiej dotyczy model projektowanego systemu, będą wykorzystywane w procesie projektowania poprzez system automatycznie wykonujący krok „uzupełnienie liczebności dla asocjacji". W artykule zaprezentowano algorytm, który na podstawie odszukiwanych informacji ze zbiorów ontologicznych automatyzuje analizowany krok.

2. Uzupełnienie liczebności dla asocjacji w diagramie modelu typów

Dla kroku „uzupełnienie liczebności dla asocjacji" zdefiniowano algorytm przedstawiony na rysunku 4.



Rys. 4. Algorytm uzupełnienia liczebności dla asocjacji
Fig. 4. Completion of multiplicity for the association algorithm

Głównym celem algorytmu jest uzupełnienie liczebności dla asocjacji, których nie zdefiniowano w procesie zdobywania wymagać. W ramach algorytmizacji należało opracować metodę automatycznego zdobywania brakującego zbioru faktów, bez którego niemożliwe jest automatyczne uzupełnienie liczebności dla asocjacji. Po analizie artefaktów wejściowych dla kilku przykładowych projektów realizowanych z wykorzystaniem metodyki CD, stwierdzono, iż nie znajdują się w nich wymagane informacje, które można by było wykorzystać w procesie automatyzacji i uzupełnienia brakujących liczebności. Dlatego należało opracować lub wykorzystać istniejące mechanizmy, pozwalające odwzo-

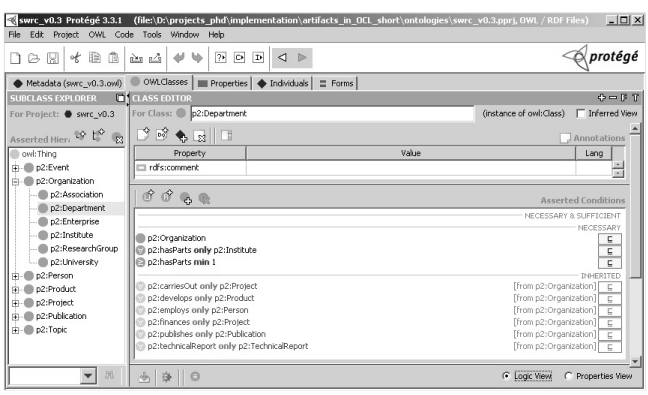
rować wiedzę i doświadczenia projektanta, który w klasycznym procesie projektowania byłby zobligowany do wykonania analizowanego kroku. Badania zostały ograniczone tylko do poszukiwania mechanizmu odwzorowującego zbiór wiedzy, który umożliwiłyby opracowanie metody odnajdującej brakujące liczebności dla asocjacji.

W celu doprecyzowania opisu, algorytm został przedstawiony w formie pseudojęzyka (rys. 5). W ramach pseudojęzyka przy użyciu instrukcji `INPUT` i `OUTPUT` przypisane są odpowiednio: dane wejściowe oraz dane wyjściowe dla opisywanego algorytmu. Instrukcja `FOREACH` jest pętlą wykonywaną na wszystkich elementach znajdujących się w liście. Ostatnia instrukcja `IF` *warunek* `czynność_1 ELSE czynność_2` służy do definiowania warunków. Jeżeli *warunek* jest spełniony, wykonywana jest `czynność_1`, w przeciwnym przypadku wykonywana jest `czynność_2`. Instrukcje warunkowe mogą być zagnieżdżane. Wykonywane w ramach algorytmu czynności zostały ponumerowane. Czynności zostały zapisane w języku naturalnym. Dla czytelności zapisu wprowadzono bloki, które ograniczone są przy użyciu znaków `{, }`.

```
INPUT: Początkowy Diagram Modelu Typów, Zbiór Modeli
Ontologicznych
OUTPUT: Diagram Modelu Typów
1) Znajdź zbiór asocjacji bez zdefiniowanych liczeb-
ności
FOREACH (asocjacja){
2) Znajdź typy znajdujące się na końcach asocjacji
w Zbiorze Modeli Ontologicznych
IF (typy posiadają zdefiniowane powiązanie w ramach
modelu ontologicznego){
IF (nazwa roli w powiązaniu odpowiada nazwie roli
w asocjacji ){
IF (powiązanie definiuje liczebności){
3) Przypisz liczebności z powiązania do asocjacji
} ELSE { 4) Przypisz liczebność '*' dla asocjacji}
} ELSE { 4) Przypisz liczebność '*' dla asocjacji}
} ELSE { 4) Przypisz liczebność '*' dla asocjacji}
}
```

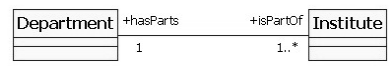
Rys. 5. Algorytm uzupełnienia liczebności dla asocjacji
Fig. 5. Completion of multiplicity for the association algorithm

Zbiory ontologiczne zostały z powodzeniem użyte do tworzenia diagramów związków encji [11]. Dlatego ramach prowadzonych badań postanowiono wykorzystać zbiory modeli ontologicznych, w procesie wyszukiwania liczebności. Dzięki temu, iż ontologia jest sposobem formalizacji zbioru wiedzy z wybranej domeny problemu [4]. W ramach opracowanej metody wykorzystano jedną z własności ontologii: możliwość definiowania powiązań pomiędzy bytami opisanymi przez klasy. Przykładowy fragment modelu ontologicznego wykorzystywanego w ramach jednego z projektów został przedstawiony na rysunku 6.



Rys. 6. Przykładowy model ontologiczny przedstawiający liczebności dla powiązania pomiędzy Wydziałem a Instytutem
Fig. 6. Exemplary ontology model describing multiplicities for the association relationship between Department and Institute

W pierwszej kolejności algorytm tworzy zbiór asocjacji z PDMT nieposiadających zdefiniowanej liczebności. Z każdej asocjacji ze zbioru, pobierane są nazwy typów (klas ze stereotypem `<<type>>`) znajdujących się na końcach asocjacji. Dla każdej odnalezionej pary typów, odszukiwane są odpowiedniki w zbiorze klas ze zbioru modeli ontologicznych, poprzez porównanie nazw typów i klas. Dla odnalezionych klas, następuje sprawdzenie, czy nazwa roli w powiązaniu pomiędzy klasami odpowiada nazwie roli w asocjacji. W przypadku odnalezienia takiej samej roli, jeżeli powiązanie posiada zdefiniowaną liczebność, następuje zastosowanie odnalezionej liczebności dla asocjacji. W każdym przypadku, gdy liczebność nie jest odnaleziona, system automatycznie przypisuje liczebność „dowolnie wiele", czyli `0..*` lub w skrócie `*`. Na rysunku 7 przedstawiono liczebności dla powiązania pomiędzy Wydziałem a Instytutem, które zostały odwzorowane na diagramie klas w języku UML na podstawie zaprezentowanego na rysunku 6 modelu ontologicznego, dzięki zastosowaniu zaprezentowanego algorytmu.



Rys. 7. Diagram klas przedstawiający liczebności dla powiązania pomiędzy Wydziałem a Instytutem
Fig. 7. Class diagram describing multiplicities for the association relationship between Department and Institute

3. Wnioski

Zaprezentowane w artykule przykładowe użycie zbiorów ontologicznych, jako uzupełnienia zbioru wiedzy niezbędnego w procesie automatyzacji, może posłużyć jako studium wyjściowe do dalszych badań nad wykorzystaniem i formalizacją zbiorów wiedzy, które można wykorzystać w procesie automatyzacji innych metodyk wytwarzania oprogramowania. Zaprezentowany algorytm zostały wykorzystany w ramach zaimplementowanego prototypu narzędzia typu CASE [10] o nazwie wPP, wspomagającego pierwszy etap projektowania wykonany zgodnie z metodyką CD.

4. Literatura

[1] J. Chessman, J. Daniels: Komponenty w UML, Wydawnictwa Naukowo-Techniczne 2004.
[2] T. Gratkowski: Rozszerzenie SPEM precyzujące definicję zadań wykonywanych w krokach, Metody i narzędzia wytwarzania oprogramowania 2007, 2007.
[3] T. Gratkowski: Model przyjętej formalizacji artefaktów i reguł dla metodyki Cheesmana-Danielsa, Pro Dialog nr 22, Wydawnictwo Nakom, Poznań 2007, strony 81-98.
[4] Thomas R. Gruber: A translation approach to portable ontology specifications. Knowl. Acquis., 5(2):199–220, June 1993.
[5] M. R Genesereth i R. E Fikes, Knowledge Interchange Format, Ver-sion 3.0 Reference Manual, Computer Science Department, Stanford University, Technical Report Logic-92-1, 1992.
[6] H. Delugach: ISO/IEC WD 24707 Information technology – Common Logic (CL) – A Framework for a Family of Logic-Based Languages. Pacific Northwest National Laboratory, Chantilly, VA, 7 June 2004.
[7] Smith, Michael K.; Chris Welty, Deborah L. McGuinness: OWL Web Ontology Language Guide, (2004-02-10). W3C.
[8] OMG: Unified Modeling Language (OMG UML), Superstructure, V2.1.2. OMG, November 2007.
[9] OMG: Object Constraint Language. OMG, version 2.0, May 2006.
[10] Definicja CASE z Carnegie Mellon Software Engineering Institute, http://www.sei.cmu.edu/legacy/case/case_whatis.html.
[11] Storey V., Sugumaran V.: The Role of Domain Ontologies in Database Design: An Ontology Management and Conceptual Modeling Environment, ACM Transactions on Database Systems, September 2006).

Artykuł recenzowany