Tomasz GRATKOWSKI
UNIWERSYTET ZIELONOGÓRSKI, INSTYTUT INFORMATYKI I ELEKTRONIKI

Formalizacja kroków dla czynności utworzenia modeli typów z metodyki Cheesmana-Danielsa na potrzeby systemu regułowego

Mgr inż. Tomasz GRATKOWSKI

Skończył studia na Wydziale Elektrycznym Politechniki Zielonogórskiej, kierunek Elektrotechnika, specjalność Inżynieria Systemów Komputerowych. Jest pracownikiem Instytutu Informatyki i Elektroniki Uniwersytetu Zielonogórskiego oraz współpracuje z firmami z branży informatycznej. Głównym tematem zainteresowań naukowych są: inżynieria oprogramowania, metodyki projektowania systemów informatycznych, język UML, wykorzystanie systemów regułowych w inżynierii oprogramowania oraz systemy rozproszone i systemy wielowarstwowe.
e-mail: T.Gratkowski@iie.uz.zgora.pl

Streszczenie

W artykule przedstawiono wykorzystanie języka deklaracyjnego w celu formalizacji kroków wykonywanych w ramach czynności utworzenia modeli typów z metodyki Cheesmana-Danielsa. Zastosowanie języka deklaracyjnego umożliwiło wykorzystanie opracowanych formalizmów w procesie budowania regułowego modelu wiedzy. Opracowane reguły zostaną wykorzystane na potrzeby projektowanego systemu regułowego wspierającego analityka systemowego w procesie projektowania systemu informatycznego. Ocena jakości modeli generowanych przez system regułowy, zostanie zweryfikowana poprzez pomiar i analizę otrzymanych wyników przy użyciu miar oprogramowania.

Formal representation of steps from 'develop type model' activity from Cheesman-Daniels methodology used in process implementation rule-base system

Abstract

Declarative programming can be used as a language for formalizing steps in 'develop type model' activity used in Cheesman-Daniels methodology. Developed formalism enables to use described rules for steps and can be used in constructing rule-base knowledge model. Described rules will be use in a rule-base system which will support a designer in process of designing the application. Quality control generated model by a rule-base system, will be verified through measurements and analysis received results apply a software metrics.

1. Wstęp

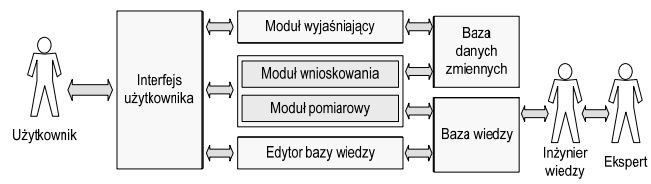
Projektowanie i implementowanie systemów informatycznych z wykorzystaniem technologii komponentowych [1] jest kolejnym etapem w procesie zgłębiania wiedzy nad efektywniejszym sposobem budowania rozległych systemów informatycznych. Z powodu złożoności zadań wykonywanych w procesie budowy systemów informatycznych, opracowano również szereg metodyk [2] wspomagających grupy informatyków. Jedną z opracowanych metodyk jest metodyka Rational Unified Proces (RUP) [3], która powstała z myślą o systemach implementowanych w technologiach komponentowych. Rozwinięciem metodyk RUP i Catalysis [4] jest metodyka zaproponowana przez J. Cheesman i J. Daniels (metodyka Cheesmana-Danielsa, dalej metodyka CD) [5], skupiająca się na specyfikacji komponentów z wykorzystaniem modeli UML [6]. Metodyka CD została podzielona na sześć dyscyplin. Każda dyscyplina składa się z etapów, w skład których wchodzą czynności, które podzielone są na kroki. W procesie przepływu czynności, analityk systemowy wykonujący poszczególne zdefiniowane w czynnościach kroki operuje na artefaktach [3]. Artefakty są efektem wykonania poszczególnych kroków oraz źródłem niezbędnych informacji dla wykonania kroków. Każdy krok można opisać ogólnym wzorem:

$$f: ArtWe \rightarrow ArtWy \tag{1}$$

gdzie: *ArtWe* – zbiór artefaktów wejściowych; *ArtWy* – zbiór artefaktów wyjściowych.

W klasycznym podejściu funkcja przejścia *f* realizowana jest poprzez analityka systemowego. Ponieważ Cheesman i Daniels nie definiują formalnych zależności pomiędzy artefaktami, proces wyodrębnienia artefaktów wyjściowych oparty jest na wiedzy i doświadczeniu projektanta, oraz na zewnętrznych źródłach niezbędnych informacji.

Prezentowany artykuł przedstawia sposób formalizacji funkcji przejścia *f* dla kroku wykonywanego w czynności 'utworzenia modeli typów' z zastosowaniem języka deklaracyjnego [7]. Formalizacja posłuży do zdefiniowania modelu wiedzy dla systemu regułowego, którego zadaniem będzie wsparcie analityka systemowego w procesie projektowanie systemu [8]. Efektem działania systemu regułowego będą propozycje artefaktów wyjściowych. Ponieważ zadaniem projektowanego systemu będzie automatyzacja zadań wykonywanych przez analityka, system będzie posiadał moduł pomiarowy w celu weryfikacji przeprowadzonych działań. Otrzymane wyniki pomiarów zostaną przeanalizowane zgodnie z wybranymi miarami oprogramowania [9]. Na rysunku 1 przedstawiono schemat modułów projektowanego systemu regułowego.



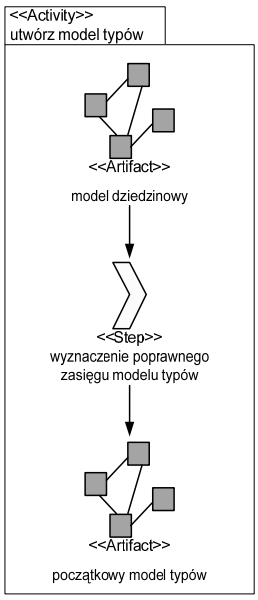
Rys. 1. Schemat blokowy zaprojektowanego systemu regułowego
Fig. 1. Block diagram designed rule-base system

2. Formalizacja kroków z czynności utworzenia modeli typów w metodyce Cheesmana-Danielsa

W metodyce CD główny nacisk położono na dyscyplinę specyfikowanie. Dyscyplina specyfikowanie została podzielona na trzy etapy. Pierwszym etapem jest identyfikowanie komponentów, w ramach którego pierwszą wykonywaną przez analityka systemowego czynnością jest utworzenie modelu typów (rys. 2). Jedynym artefaktem wejściowym jest model dziedzinowy. Model dziedzinowy służy do zdefiniowania pojęć użytych podczas budowy systemu informatycznego. Definiowane pojęcia pochodzą z konkretnej dziedziny lub dziedzin, dla której projektowany jest system informatyczny. Model dziedzinowy jest przedstawiany przy użyciu diagramów klas. Artefaktem wyjściowym jest początkowy model typów. Początkowy model typów jest modelem specyfikacji. Tworzony jest również przy użyciu diagramów klas, a wiele klas posiada takie same nazwy jak klasy z modelu dziedzinowego. Jednak klasy w modelu typów przedstawiają systemową reprezentację wybranego pojęcia, rozpatrując je bardziej dokładnie i bardziej precyzyjnie definiując.

W ramach czynności utworzenie modelu typów wykonywany jest jeden krok „wyznaczenie poprawnego zasięgu modelu typów" (dalej WPZMT). Główne zadanie, jakie postawiono analitykowi w ramach kroku WPZMT, jest utworzenie na bazie modelu dziedzinowego początkowego modelu typów.

Na początku kroku następuje przekopiowanie modelu pojęciowego a następnie dodanie do niego lub usunięcie z niego klas reprezentujących typy, w celu zdefiniowania poprawnego zasięgu modelu. Głównym kryterium podczas dodawania lub usuwania klas jest informacja czy klasa opisana w modelu pojęciowym przedsiębiorstwa ma być przechowywana przez projektowany system.



Rys. 2. Diagram aktywności opisujący czynność tworzenia modelu typów
Fig. 2. 'Develop type model' activity described in activity diagram

Przyjęto, iż każdy krok realizowany w ramach czynności, jest opisany ogólnym wzorem 1. Aby zrealizować opisaną funkcję dla kroku WPZMT, która przekształca artefakt wejściowy na artefakt wyjściowy należy wykonać przekształcenie zgodnie z algorytmem 1.

```
INPUT: ArtefaktWejściowy

OUTPUT: ArtefaktWyjściowy


IF ArtefaktWejściowy IS modelem dziedzinowym (reguła1)

THEN

    1) przekopiuj ArtefaktWejściowy i oznacz jako
    ArtefaktWyjściowy

    2) uzupełnij klasy w ArtefaktWyjściowy o potrzeb-
    ne atrybuty i typy

    3) uściślij i uzupełnij w ArtefaktWyjściowy po-
    wiązania

    4) zdefiniuj w ArtefaktWyjściowy poprawny zasięg

    5) wykonaj regułę 4

END IF
```

Algorytm 1. Algorytm opisujący funkcję przejścia dla czynności WPZMT

Przedstawiony algorytm poza określonymi zadaniami 1-4, wykorzystuje również reguły 1 i 4, opisane przy użyciu deklaracyjnej formuły IF <warunek> THEN <wynik>. Taka forma zapisu wskazuje *co jest* istotą oprogramowanego logicznego warunku i jest przeciwieństwem dla języków imperatywnych, gdzie istotne jest *jak*.

Reguła 1 opisuje warunki, jakie musi spełnić artefakt wejściowy. Dla kroku WPZMT artefakt wejściowy mysi być modelem dziedzinowym, dlatego reguła 1 sprawdza wszystkie własności, jakie musi spełniać model dziedzinowy. Reguła 2 analizuje własności, jakie musi posiadać artefakt wyjściowy, czyli model o poprawnym zasięgu. Kolejna reguła numer 3 sprawdza czy artefakt wejściowy jest przodkiem artefaktu wyjściowego. Opisane reguły 1-3 są wykorzystywane w procesie wnioskowania, jaki realizowany jest w regule 4.

```
IF
        ArtefaktWejściowy definiuje pojęcia
        z dziedziny implementowanego problemu
    AND
        ArtefaktWejściowy definiuje związki
    AND
        ArtefaktWejściowy jest diagramem klas
THEN
    ArtefaktWejściowy modelem dziedzinowym
```

Reguła 1. Model dziedzinowy

```
IF
        ArtefaktWyjściowy definiuje jedynie te
        pojęcia z dziedziny implementowanego
        problemu, które zostaną
        zaimplementowane przez system,
    AND
        ArtefaktWejściowy definiuje wszystkie
        związki zachodzące pomiędzy bytami
THEN
    ArtefaktWyjściowy jest modelem o poprawnym
    zasięgu
```

Reguła 2. Model o poprawnym zasięgu

```
IF
        ArtefaktWejściowy jest kopią
        ArtefaktWyjściowy
THEN
    ArtefaktWejściowy jest przodkiem
    ArtefaktWyjściowy
```

Reguła 3. Przodek

```
IF
        ArtefaktWejściowy jest modelem
        dziedzinowym
    AND
        ArtefaktWejściowy jest przodkiem
        ArtefaktWyjściowy
    AND
        ArtefaktWyjściowy jest modelem
        o poprawnym zasięgu
THEN
    ArtefaktWyjściowy jest początkowym
    modelem typów
```

Reguła 4. Początkowy model typów

W przedstawionych regułach zdefiniowano warunki, których sprawdzenie i rozwiązanie wymaga przeprowadzenia pomiaru i analizy ArtefaktuWejściowego oraz ArtefaktuWyjściowego. Analiza przeprowadzona zostanie na podstawie wybranych miar [10], które zostaną odpowiednio dopasowane do specyfiki analizowanych artefaktów.

3. Implementacja

Przedstawione w rozdziale 2 reguły zostaną zaimplementowane przy użyciu środowiska regułowego Drools [11]. Środowisko Drools implementuje algorytm Rete autorstwa Charles'a Forgy [12]. W Drools wprowadzono mechanizm Semantics Module Framework (SMF) wykorzystujący domain specific rule languages (DSL) [13].

DSL jest abstrakcyjnym językiem wyższego rzędu, umożliwiającym zoptymalizowanie przygotowywanych reguł dla dziedziny rozwiązywanego problemu.

Rysunek 3 przedstawia zapis reguły 'model dziedzinowy' zapisanej w pliku XML dla środowiska Drools. Zaprezentowany przykład wykorzystuje implementację języka Java dla Drools.

```
<?xml version="1.0" encoding="UTF-8"?>

<!--     <<szczegoly przeplywu czynnosci>>
         identyfikowanie_komponentow

         <<czynnosc>>
         1_1-utworz_model_typow -->

<rule-set name="utworz_model_typow"

       xmlns="http://drools.org/rules"

       xmlns:java="http://drools.org/semantics/java"

       xmlns:xs="http://www.w3.org/2001/XMLSchema-
       instance"

       xs:schemaLocation="http://drools.org/rules
       rules.xsd
       http://drools.org/semantics/java java.xsd">


       <iport>org.uz.iie.cd_methodology.profile.
       comonent_identyfikaction.
         develop_type_model.DomainModel</import>

       <iport>org.uz.iie.cd_methodology.profile.
       Artifact</import>

      <rule name="Domain Model" salience="10">

         <parameter identifier="domainModel">

            <class>DomainModel</class>

         </parameter>

         <java:condition>

            <!-- czy definiuje pojecia z dziedziny
                 implementowanego problemu -->

            DomainModel.doDefineNotionsFrom
            ProblemDomain() == true

         </java:condition>

         <java:condition>

            <!-- czy jest diagramem klas -->

            domainModel.doClassDiagram() == true

         </java:condition>

         <java:condition>

            <!-- czy definiuje zwiazki pomiedzy
                 klasami -->

            domainModel.doDefineRelations() == true

         </java:condition>

         <java:consequence>

             Artifact.printType
             (domainModel.getOutput());

         </java:consequence>

      </rule>

</rule-set>
```

Rys. 3. Definicja reguły 'model dziedzinowy' dla środowiska Drools
Fig. 3. 'Domain model' rule definition for Drools a rules engine

Zastosowanie modelu, gdzie reguły mogą być opisane jako pliki XML upraszcza wprowadzanie nowych reguł oraz poprawianie i modyfikację już istniejących. Dodatkowo zaimplementowany SMF dla języka Java umożliwia bezpośrednie wykorzystywanie metod z języka Java wewnątrz opisu reguł.

4. Wnioski

Poniższy artykuł prezentuje na przykładzie wybranego kroku, sposób formalizacji funkcji przejścia realizowanych w krokach przyporządkowanych czynnościom realizowanym w metodyce CD. Przedstawione rozwiązanie będzie wykorzystane w celu sformalizowania wszystkich kroków, jakie realizowane są w ramach całej metodyki CD. Umożliwi to przeprowadzenie analizy i wyboru kroków, które mogą być w całości zautomatyzowane. Proces automatyzacji realizowany będzie przy użyciu systemu regułowego, który poprzez wykorzystanie dodatkowego modułu pomiarowego i zastosowaniu analizy opartej o miary oprogramowania, będzie analizował jakość otrzymanych modeli. Dodatkowa analiza będzie rozszerzała model wiedzy, który system regułowy może wykorzystać podczas wykonywania kolejnych przekształceń. Proponowana formalizacja umożliwi również przeprowadzenie dalszych badań nad czynnościami, które na bazie dzisiejszego stanu wiedzy nie mogą być zautomatyzowane.

5. Literatura

[1] G. T. Heineman, W. T. Councill: Component-base software engineering: Putting the Pieces Together, Addison-Wesley, 2001.

[2] T. Gratkowski: Cykl życia projektu informatycznego, Pierwsza Konferencja Naukowa 'Informatyka - sztuka czy rzemiosło', strony 113–118, Instytut Informatyki i Elektroniki, Uniwersytet Zielonogórski, Oficyna Wydawnicza Uniwersytetu Zielonogórskiego, 2004.

[3] P. Kruchten: The Rational Unified Process An Introdaction, Addison-Wesley, 2000.

[4] D. F. D'Souza, A. C. Wills: Objects, Components, and Frameworks with UML: The Ca-talysis Approach, Addison-Wesley, 1998.

[5] J. Chessman, J. Daniels: Komponenty w UML, Wydawnictwa Naukowo-Techniczne, Warszawa, 2004.

[6] OMG: Unified Modeling Language: Superstructure, v2.0, Object Management Group, July 2005.

[7] R. G. Ross: Principles of the Business Rule Approach, Addison Wesley, 2003

[8] T. Gratkowski: Automatyczne generowanie artefaktów wyjściowych dla czynności określenia interfejsów i operacji systemowych w metodzie Cheesmana-Danielsa z wykorzystaniem systemu ekspertowego, Computer Methods and Systems - CMS '05, strony 483-488, Oprogramowanie Naukowo-Techniczne - Vol. 2, 2005.

[9] pod redakcją J. Górski: Inżynieria oprogramowania w projekcie informatycznym, MIKOM, 2000.

[10] M. Genero, M. Piattini, C. Calero: A Survey of Metrics for UML Class Diagrams, Journal of Object Technology, Vol. 4, No. 9, November-December 2005, pp. 59-92.

[11] Mark Proctor: Business Rules Engines, Business Rules at Javapolis 2005, http://javapolis.com/confluence/display/JP05/Business+Rules+Engines.

[12] C. Forgy: Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem, Artificial Intelligence, 19, pp 17-37, 1982.

[13] Strona organizacji Drools: www.drools.org.

Artykuł recenzowany