

# Ograniczenia dynamiczne w modelowaniu zachowania obiektów

Zygmunt Mazur, Artur Wilczek

*Politechnika Wroclawska Wydziałowy Zakład Informatyki, Wydział Informatyki i Zarządzania*

{zmazur, wilczek}@ci.pwr.wroc.pl

## Streszczenie

Jednym z kluczowych etapów analizy i projektowania systemu w metodologiach obiektowych jest stworzenie modelu zachowania. UML wspiera modelowanie zachowania obiektów szeregiem dostępnych notacji. Ich przykładem mogą być diagramy stanów wzorowane na mapach stanów Harela [HAREL1985]. Diagramy stanów wykorzystywane są między innymi do opisu zachowania obiektów poszczególnych klas. Rozdział przedstawia zastosowanie języka ODCL jako narzędzia specyfikacji ograniczeń dynamicznych w modelu zachowania obiektów. Proponuje także algorytmy konstrukcji diagramów stanów na podstawie specyfikacji ograniczeń dynamicznych.

## 1 Wprowadzenie

Stworzenie poprawnego modelu zachowania obiektów ma kluczowe znaczenie dla powodzenia projektu, jest jednym z podstawowych etapów tworzenia systemu informatycznego przy wykorzystaniu metodologii obiektowych.

Model zachowania (działania, dynamiki) obiektów zawiera opis tych aspektów systemu, które dotyczą realizowanych w systemie operacji oraz ich kolejności. W modelu zachowania ujęte są również zdarzenia występujące w systemie oraz wzajemne interakcje obiektów. Przy opracowaniu modelu zachowania wykorzystać można diagramy interakcji, diagramy stanów i aktywności.

Diagramy stanów opisują w formie graficznej zbiór stanów oraz przejść pomiędzy nimi, zdarzeń i wzajemnych interakcji obiektów. Diagramy stanów wzorowane są na mapach stanów Harela [HAREL1985]. Mapy stanów są z kolei rozszerzeniem notacji diagramów automatów skończonego stanów. Rozszerzenie to obejmuje wprowadzenie hierarchii stanów, równoległości procesów oraz przekazywanie komunikatów.

Praca [MAWI2001] proponuje metodę specyfikacji dynamicznych więzów integralności w obiektowym modelu danych. W niniejszym rozdziale wykorzystamy to rozwiązanie do specyfikacji ograniczeń dynamicznych i konstrukcji na ich podstawie diagramów stanów modelujących poprawne zachowanie obiektów.

Dynamiczne więzy integralności pozwalają na stawianie ograniczeń ponad sekwen-

cją, zwykle uporządkowanych w czasie stanów bazy danych, dotyczą więc przede wszystkim obiektów trwałych. Przez ograniczenia dynamiczne dla obiektów rozumiemy warunki poprawności operacji realizowanych na obiektach odwołujące się nie tylko do pojedynczego stanu obiektów, lecz do sekwencji stanów.

## 2 Ograniczenia dynamiczne

Wiele ograniczeń, często opisanych w języku naturalnym, nie znajduje reprezentacji w standardowych narzędziach specyfikacji. Zaniechanie ich specyfikacji może prowadzić do niejednoznaczności modelu, co w konsekwencji powoduje błędy w działaniu systemu. Istnieje szereg formalnych języków specyfikacji więzów integralności. Większość z nich wymaga jednak znajomości zaawansowanych teorii matematycznych lub logicznych co znacznie ogranicza ich dostępność. Język OCL (ang. *Object Constraint Language*) powstał w celu wypełnienia tej luki. Pozostając językiem formalnym jest czytelny i prosty w użyciu [OCL2001].

### 2.1 Język OCL

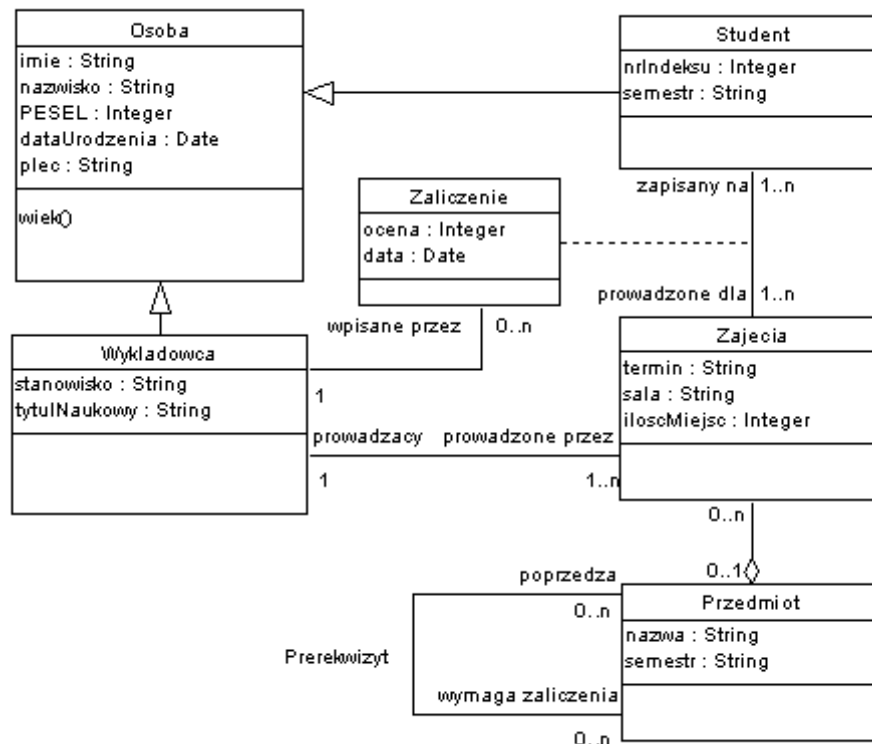
Język OCL jest językiem opisu ograniczeń w modelu obiektowym. Znalazł zastosowanie przy specyfikacji inwariantów klas, tworzeniu specyfikacji metod klas, warunków przejścia pomiędzy stanami w diagramach stanów. Przy wykorzystaniu języka OCL można specyfikować formuły odwołujące się do wszystkich elementów wykorzystywanych w modelowaniu statycznym systemu tj. klas, związków, atrybutów klas, metod. OCL można wykorzystać do specyfikacji warunków *pre* i *post* dla operacji i metod. W warunkach *post* odwoływać można się do wartości właściwości klasy przed i po wykonaniu metody bądź operacji. Każda formuła języka OCL ma swój typ, wymagana jest zgodność typów, dostępne są także typy predefiniowane np. kolekcje udostępniające sporo użytecznych własności i operacji.

Formuły OCL specyfikują właściwości instancji wskazanego typu. Jeżeli formuły nie stanowią opisu diagramów, z których jednoznacznie wynika jakiego typu dotyczą, wymagane jest wskazanie kontekstu formuły. Deklaracja kontekstu formuły następuje po słowie kluczowym *context*, po którym wskazać należy nazwę odpowiedniego typu lub wybraną metodę typu. Wyróżniono trzy stereotypy formuł OCL. Wskazanie stereotypu formuły musi być umieszczone bezpośrednio po deklaracji jej kontekstu i następuje poprzez słowa kluczowe oznaczające: **inv** – inwarianty klasy (w kontekście typu), **pre**, **post** – warunki poprawności dla metod i operacji klas (w kontekście metody bądź operacji typu).

Nawigacja poprzez wszystkie dostępne w statycznym modelu systemu klasy, związki, atrybuty i metody następuje poprzez notacje kropkową. Mechanizm ten umożliwia odwołanie się do wszystkich obiektów powiązanych z instancją kontekstową. Poczynając od wybranego obiektu możemy nawigować do powiązanych obiektów i ich własności wskazując etykiety asocjacji powiązanych klas. Predefiniowane typy zbiorowe OCL oferują szereg predefiniowanych właściwości i operacji. Należą do nich między innymi własności: **notEmpty** - typu logicznego, spełniona, jeżeli zbiór jest pusty, **size** – ilość

elementów zbioru. Dostęp do predefiniowanych własności bądź operacji na kolekcjach uzyskuje się przy użyciu symbolu „->”.

Jeżeli wystąpi potrzeba odwołania się do kolekcji wszystkich instancji danego typu (wszystkich obiektów danej klasy) wówczas możemy wykorzystać predefiniowaną własność **allInstances**. Należy pamiętać, iż właściwość ta nie jest określona dla niektórych predefiniowanych typów np.: **Real**, **Integer**. W przykładzie (2) wykorzystano kolejną predefiniowaną operację na kolekcjach **forall** – służącą specyfikacji warunków logicznych, które muszą być spełnione dla wszystkich elementów kolekcji.



Rysunek 1. Przykładowa hierarchia klas

Dla przykładowej hierarchii klas przedstawionej na rysunku 1., proste ograniczenie wyrażone w języku naturalnym zdaniem „wykładowcą może być wyłącznie osoba pełnoletnia” można zapisać w OCL wyrażeniem postaci:

```
(1) context Wykładowca inv :
    self.wiek() > 18
```

W kolejnym przykładzie zastosowano dostępne w języku OCL operacje na kolekcjach. Niech dane będzie ograniczenie: „na zajęcia nie może zapisać się więcej studentów niż maksymalna liczba studentów określona dla poszczególnych zajęć”, jego specyfikację w języku OCL można przedstawić następująco:

```
(2) context Zajecia inv MaxIloscStudentow
    zajecia.allInstances -> forall( z |
        z.zapisany_na -> notEmpty implies
        z.zapisany_na ->size <= z.iloscMiejsc)
```

W powyższym przykładzie wyrażenie `z.zapisany_na` jest przykładem nawigacji poprzez związki między obiektami, dla poszczególnych zajęć oznacza zbiór studentów, którzy są na dane zajęcia zapisani.

## 2.2 Język ODCL

Język OCL umożliwia specyfikację statycznych ograniczeń, jest jednak niewystarczający przy specyfikacji ograniczeń dynamicznych. Aby umożliwić specyfikację ograniczeń dynamicznych, proponujemy rozszerzenie języka OCL o operatory logiki temporalnej, wykorzystywane już z powodzeniem do specyfikacji dynamicznych więzów integralności w bazach danych [CHOM1992], [SALI1988]. Otrzymany język umożliwiający specyfikację dynamicznych ograniczeń, w modelu obiektowym, proponujemy nazwać od nazwy pierwowzoru: ODCL (ang. *Object Dynamic Constraint Language*). Połączenie języka OCL i logiki temporalnej umożliwia tworzenie formalnych opisów zachowania systemu przez określenie poprawnych sekwencji stanów, co jest szczególnie użyteczne przy modelowaniu zachowania systemu.

Nasza propozycja obejmuje formułowanie ograniczeń ponad skończonymi sekwencjami stanów obiektów na etapie modelowania systemu, a nie w trakcie jego eksploatacji. W związku z tym wprowadzenie zarówno operatorów czasu przeszłego jak i przyszłego nie zwiększyłyby ekspresywności proponowanego języka [EMER1990], [KLIM1999]. Prace z dziedziny dynamicznych więzów integralności w bazach danych nie dają w tej kwestii jednoznacznych rozstrzygnięć, proponują wykorzystanie logiki czasu przyszłego [SALI1988], [LIZH1991], [GELI1996] bądź logiki czasu przeszłego [CHOM1992], [SASCH1992], [CHOM1995]. Obecne w literaturze przekonanie o bardziej naturalnym formułowaniu ograniczeń w logice temporalnej czasu przeszłego [LPZ1985] skłania nas do zaproponowania rozszerzenia języka OCL wyłącznie o zestaw operatorów temporalnych czasu przeszłego. Dodatkowym argumentem za wprowadzeniem operatorów czasu przeszłego jest obecność operatora `@pre` w języku OCL pozwalającego na odwołanie się do poprzedniej wartości własności obiektów w warunkach post dla operacji. Język OCL nie przewiduje możliwości wykorzystanie operatora umożliwiającego odwołanie się do przyszłych wartości własności obiektów.

W języku ODCL, oprócz standardowych konstrukcji języka OCL, wykorzystywać można także operatory temporalne czasu przeszłego interpretowane następująco:

- ***prev* A** - jest spełnione w stanie aktualnym, jeżeli A jest spełnione w poprzednim stanie.
- **A since B** - jest spełnione w aktualnym stanie, jeżeli istniał stan wcześniejszy, w którym spełnione było B.
- **sometime A** - kiedyś w przeszłości spełnione było A.
- **always A** - zawsze w przeszłości spełnione było A.

gdzie: A, B są wyrażeniami logicznymi języka ODCL. Poprzestaniemy tutaj na takim nieformalnym opisie semantyki operatorów temporalnych. Formalną definicję składni i semantyki języka ODCL znaleźć można w pracy [WILCZ2003]. Pełne przedstawienie własności logiki temporalnej znaleźć można w pracy [KLIM1999].

Przy wykorzystaniu języka ODCL można specyfikować formuły odwołujące się do wszystkich elementów, które dostępne są w języku OCL, ze statycznego modelu systemu tj. klas, związków, atrybutów klas, metod. Język ODCL można wykorzystać do specyfikacji inwariantów klas a także warunków pre i post dla operacji i metod.

Dla przykładowej hierarchii klas przedstawionej na rysunku 1. ograniczenie dynamiczne wyrażone w języku naturalnym zdaniem „zanim student zapisze się na zajęcia z przedmiotu Analiza II musi zaliczyć przedmiot Analiza I ” można zapisać w ODCL wyrażeniem postaci:

```
(3) context Student inv Analiza:
    let: zapisany(p String):Boolean=
        self.prowadzone_dla->exists(z Zajecia|
            z.przedmiot.nazwa=p)
    let: zaliczony(p String):Boolean=
        self.zaliczenie->exists(z Zaliczenie|
            z.zajecia.przedmiot.nazwa = p
            and z.ocena > 2) in
    zapisany('Analiza II') implies sometime zaliczony('Analiza I')
```

### 3 Modelowanie zachowania systemu

Język UML przewiduje możliwość tworzenia opisów zachowania systemu przy wykorzystaniu diagramów stanów. Pokażemy jak formuły ODCL służyć mogą konstruowaniu diagramów stanów reprezentujących ograniczenia dynamiczne, szczegółowe omówienie zagadnień prezentowanych tutaj znaleźć można w [WILCZ2003].

Proponowana w tym rozdziale metoda konstruowania diagramów stanów na podstawie specyfikacji dynamicznych ograniczeń w języku ODCL jest adaptacją metod przedstawionych w pracach [SALI1988], [SASCH1992].

#### 3.1 Normalizacja formuł ODCL

Wyrażenia temporalne języka ODCL specyfikujące ograniczenia dynamiczne mają postać formuł logiki temporalnej, których nietemporalnymi podformułami są wyrażenia logiczne języka OCL. Wyrażenia temporalne języka ODCL można transformować do odpowiadającej im postaci normalnej, traktując nietemporalne podformuły jak niepodzielne formuły elementarne. Bazujemy tutaj na metodach normalizacji formuł logiki temporalnej czasu przyszłego [SALI1988] oraz czasu przeszłego [SASCH1992].

Normalizacja formuł języka ODCL polega na ich sprowadzeniu do dysjunkcyjnej temporalnej postaci normalnej, czyli formuły będącej wyłącznie alternatywą podformuł postaci  $A \wedge \text{prev}(B)$ , gdzie A jest podformułą nietemporalną (wyrażeniem logicznym języka OCL) zaś B jest wyrażeniem języka ODCL.

Zwykle normalizację formuły temporalnej rozpoczynamy od wykorzystania praw logicznego rachunku zdań. Implikacja i równoważność zastępowane są odpowiadającymi im konstrukcjami wykorzystującymi alternatywę i koniunkcję. Następnie otrzymana formuła podlega dalszym transformacjom poprzez stosowanie odpowiednich reguł tem-

poralnych dla najbardziej zewnętrznego operatora temporalnego i dalej rekurencyjnie dla pozostałych operatorów, dopóki nie zostaną otoczone operatorem temporalnym *prev*. Transformacja taka prowadzi do uzyskania formuł złożonych z podformuł nietemporalnych  $A$  i podformuł temporalnych **prev**( $B$ ).

Poniższe reguły rekursji temporalnej odpowiadają prawom rekursywnej równoważności klasycznych operatorów temporalnych [KLIM1999], odzwierciedlają semantykę operatorów temporalnych języka ODCL:

$$\text{sometime } \alpha \Leftrightarrow \alpha \vee \text{prev}(\text{sometime } \alpha)$$

$$\text{always } \alpha \Leftrightarrow \alpha \wedge \text{prev}(\text{always } \alpha)$$

$$\alpha \text{ since } \varphi \Leftrightarrow \alpha \vee (\varphi \wedge \text{prev}(\alpha \text{ since } \varphi))$$

gdzie:  $\alpha$ ,  $\varphi$  jest wyrażeniem logicznym ODCL. Dowody powyższych reguł znaleźć można w pracy [WILCZ2003].

### 3.2 Algorytm konstrukcji diagramów stanów

Algorytm przedstawiony w pracy [SASCH1992] pozwala na transformację formuł logiki temporalnej czasu przeszłego na odpowiadające im diagramy przejść. Wprowadzenie modyfikacji uwzględniających semantykę operatorów temporalnych dostępnych w języku ODCL pozwala zaadoptować to podejście do konstrukcji diagramów stanów. Diagramy przejść uzyskane metodą zaproponowaną w pracy [SASCH1992] są wartościowane stopniowo poczynając od aktualnego stanu wstecz poprzez całą sekwencję stanów. Nie można w związku z tym zastosować tej metody wprost do konstrukcji diagramów stanów opisujących przejścia od stanu początkowego w przód do stanu końcowego w sekwencji stanów. Praca [SASCH1992] przewiduje zastosowanie odwróconych diagramów przejść przy weryfikacji ograniczeń w bazach danych, uzyskane po takiej modyfikacji diagramy przejść przyjmujemy za uproszony model diagramów stanów UML. Diagramy stanów można otrzymać z odwróconych diagramów przejść przyjmując, że każdy węzeł diagramu przejść odpowiada stanowi w diagramie stanów i każde przejście w diagramie przejść odpowiada przejściu pomiędzy stanami w diagramie stanów.

Aby zbudować diagram stanów należy formuły zapisane w logice temporalnej czasu przeszłego sprowadzić do postaci normalnej poprzez ich transformacje w teraźniejszą nie temporalną część do sprawdzenia w stanie aktualnym i temporalną część przeszłą z operatorem **prev**, do weryfikacji w pozostałej sekwencji stanów. Część nie temporalna służy specyfikacji warunku przejścia pomiędzy stanami w diagramach stanów (ang. *guards*).

Przy danej formule  $\alpha$  języka ODCL odpowiadający jej diagram stanów zbudować można realizując następujący algorytm:

1. rozpoczynamy od pustego diagramu stanów,
2. wstawiamy stan początkowy etykietowany formułą  $\alpha$ ,
3. dla każdego stanu  $v$  w diagramie wykonujemy transformacje jego etykiety

do dysjunkcyjnej temporalnej postaci normalnej.

1. dla każdej uzyskanej podformuły postaci:  $A \wedge \mathbf{prev}(B)$ , jeżeli nie istnieje jeszcze stan etykietowany formułą  $B$ , dodajemy nowy stan  $v'$  o etykiecie  $B$ , w przeciwnym przypadku podstawiamy pod  $v'$  znaleziony stan etykietowany formułą  $B$
2. dodajemy przejście  $(v, v')$  etykietowane formułą  $A$ .
3. jeżeli pomiędzy stanami  $v$  i  $v'$  istnieje więcej niż jedno przejście, zastępujemy je pojedynczym przejściem etykietowanym alternatywą etykiet pierwotnych przejść.

Tak zbudowany diagram stanów nie ma oznaczonych stanów końcowych. Stany końcowe to stany o etykietach danych formułami temporalnymi spełnionymi w pustej sekwencji stanów. Na podstawie semantyki operatorów temporalnych definiujemy funkcję końca  $K: \alpha \rightarrow \{\text{true}, \text{false}\}$ , którą wykorzystamy do wyznaczenia zbioru stanów końcowych. Niech  $\square, \psi, \rho$  będą formułami nietemporalnymi, wówczas:

$$\begin{aligned}
 K(\rho) &:= \rho \equiv \text{true} \\
 K(\neg\varphi) &:= \neg K(\varphi) \\
 K(\varphi \wedge \psi) &:= K(\varphi) \wedge K(\psi) \\
 K(\varphi \vee \psi) &:= K(\varphi) \vee K(\psi) \\
 K(\mathit{prev} \varphi) &:= K(\varphi) \\
 K(\mathit{always} \varphi) &:= \text{true} \\
 K(\mathit{sometime} \varphi) &:= \text{false}
 \end{aligned}$$

### 3.3 Przykład

Dla zobrazowania przedstawionego wyżej podejścia przeanalizujemy ograniczenie dane zdaniem

„zanim student otrzyma wpis zaliczenia zajęć z przedmiotu, musi zapisać się na dane zajęcia”.

Jego specyfikacja może mieć postać:

```

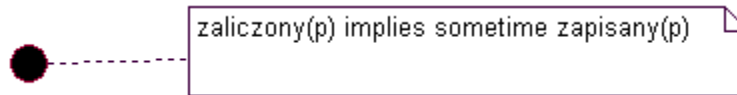
(6) context Student inv PSI:
  let: p:String = 'PSI'
  let: zapisany(n String):Boolean=
    self.prowadzone_dla->exists(z Zajecia|
      z.przedmiot.nazwa=n)
  let: zaliczony(n String):Boolean=
    self.zaliczenie->exists(z Zaliczenie|
      z.zajecia.przedmiot.nazwa=n
      and z.ocena > 2) in
  zaliczony(p) implies sometime zapisany(p)

```

Dla uproszczenia przykładu pominiemy w formule specyfikującej ograniczenie deklaracje kontekstu i wyrażeń pomocniczych ograniczając się do formuły postaci:

(7)  $\text{zaliczony}(p) \text{ implies sometime zapisany}(p)$

Zgodnie z pierwszym i drugim krokiem algorytmu rozpoczynamy od pustego diagramu stanów a następnie formułę ograniczenia dynamicznego umieszczamy jako etykietę stanu początkowego. Otrzymujemy początkowy diagram stanów (rysunek 2).



**Rysunek 2. Przykładowy diagram stanów**

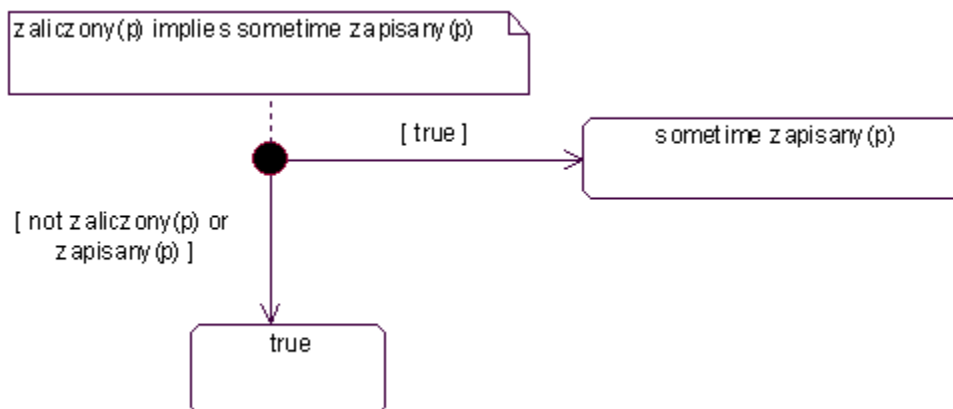
Zgodnie z trzecim krokiem algorytmu normalizujemy formułę etykietującą stan początkowy (jedyne dotychczasowe stan naszego diagramu). Rozpoczynamy od zastąpienia implikacji alternatywą zgodnie z prawem logicznego rachunku zdań i otrzymujemy formułę:

(8)  $\text{not zaliczony}(p) \text{ or sometime zapisany}(p)$

Następnie, stosując do kwantyfikatora temporalnego jedną z reguł rekursji temporalnej oraz uzupełniając formułę tautologiami, otrzymujemy wyrażenie w temporalnej dysjunkcyjnej postaci normalnej:

(9)  $\text{not zaliczony}(p) \text{ and prev}(\text{true})$   
 $\text{or zapisany}(p) \text{ and prev}(\text{true})$   
 $\text{or true and prev}(\text{sometime zapisany}(p))$

Realizując kolejne kroki algorytmu uzyskujemy kolejny diagram stanów (rysunek 3).



**Rysunek 3. Przykładowy diagram stanów**

Następnie sprowadzamy formuły etykietujące otrzymane stany do dysjunkcyjnej temporalnej postaci normalnej. Stan etykietowany zdaniem zawsze prawdziwym jest znormalizowany, pozostaje więc normalizacja formuły:

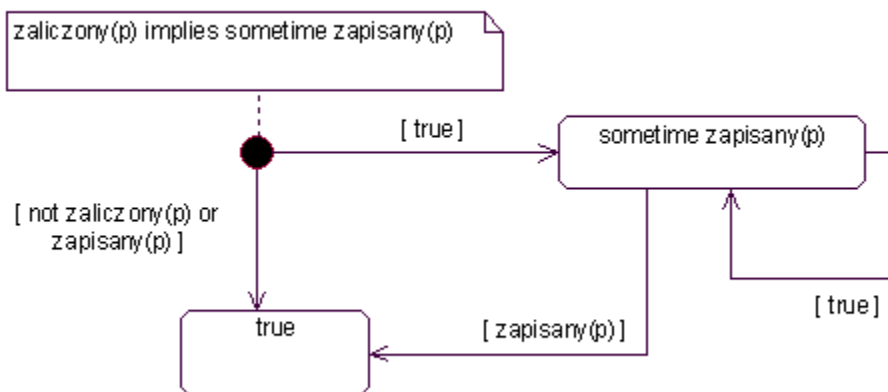
(10)  $\text{sometime zapisany}(p)$

Ponownie stosując do kwantyfikatora temporalnego jedną z reguł rekursji temporalnej oraz uzupełniając formułę tautologiami otrzymujemy wyrażenie w temporalnej dys-

junkcyjnej postaci normalnej:

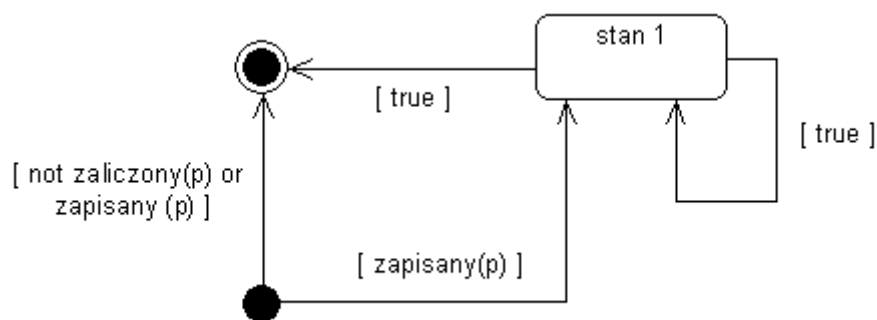
```
(11) zapisany(p) and prev(true)
    or true and prev(sometime zapisany(p))
```

Ponieważ odpowiednie stany istnieją już na diagramie, uzupełniamy wyłącznie przejścia pomiędzy stanami i uzyskujemy kolejny diagram stanów (rysunek 4).



**Rysunek 4. Przykładowy diagram stanów**

Diagram stanów uzyskany dotąd nie jest zgodny z wymaganiami UML. Aby uzyskać diagram zgodny z wymaganiami UML wykorzystujemy funkcję końca do wyznaczenia stanów końcowych. Następnie stosujemy transformacje do odwróconego diagramu przejść. Po pominięciu temporalnych etykiet stanów otrzymujemy diagram w końcowej postaci (rysunek 5).



**Rysunek 5. Przykładowy diagram stanów**

## 4 Podsumowanie

Autorzy proponują język ODCL jako metodę specyfikacji ograniczeń dynamicznych w modelu zachowania systemu. Język ODCL zachowuje czytelność i prostotę użycia konstrukcji językowych charakterystyczną dla języka OCL. Łatwość i przejrzystość narzędzi specyfikacji ograniczeń dynamicznych może przyczynić się do ich szerszego wykorzystania w modelowaniu zachowania systemów.

Zaproponowane algorytmy konstrukcji diagramów stanów na podstawie specyfikacji dynamicznych więzów integralności można wykorzystać w narzędziach wspierających modelowanie systemów obiektowych w notacji UML. Diagramy uzyskiwane w wyniku zastosowania proponowanych algorytmów po pominięciu temporalnych formuł etykietujących stany diagramu są zgodne z przyjętymi w UML notacjami.

## Bibliografia

- [CHOM1992] J Chomicki, *History-less checking of dynamic integrity constraints*, Proceedings of the Eighth International Conference on Data Engineering IEEE Computer Society , 557-564 , 1992 .
- [CHOM1995] J Chomicki, *Efficient Checking of Temporal Integrity Constraints Using Bounded History Encoding Tods*, 1995 , 149-186 .
- [EMER1990] E.A Emerson, *Temporal and Modal Logic. Handbook of Theoretical Compute Science*, Elsevier Science Publishers, vol. B, 1990, 996-1072.
- [GELI1996] M Gertz i U.W Lipeck, *Deriving Optimized Integrity Monitoring Triggers from Dynamic Integrity Constraints*, Data Knowledge Engineering, 1996, 163-193.
- [HAREL1985] D Harel i , *Statecharts, A Visual Formalism for Complex Systems*, Science of Computer Programming, 1987, 231-274.
- [KLIM1999] R Klimek, *Wprowadzenie do logiki temporalnej*, Wydawnictwa AGH, 1999 .
- [LIPZH1991] U.W Lipeck i H Zhou, *Monitoring Dynamic Integrity Constraints on Finite State Sequences and Existence Intervals*, Workshop on Foundations of Models and Languages for Data and Objects , 1991 , 115-130 .
- [LPZ1985] O Lichtenstein, A Pnueli i L.D Zuck, *The Glory of the Past Logics of Programs*, 1985 , 196-218 .
- [MAWI2001] Z Mazur i A Wilczek, *Dynamiczne więzy integralności w obiektowym modelu danych*, Katedra Automatyki Akademii Górniczo-Hutniczej w Krakowie, 2001 .
- [OCL2001] *Object Constraint Language Specification Version 1.4*, Object Management Group, 2001.
- [SALI1988] G Saake i U.W Lipeck, *Using Finite-Linear Temporal Logic for Specifying Database Dynamics*, Springer, 1988, 288-300.
- [SASCH1992] G Saake i S Schwiderski, *Monitoring Temporal Permissions Using Partially Evaluated Transition Graphs*, in *Modelling Database Dynamics*, 4th International Workshop on Foundations of Models and Languages for Data and Objects, 1992, 196-217.

- [WILCZ2003] A Wilczek, *Dynamiczne więzy integralności w obiektowym modelu danych, rozprawa doktorska Wydział Informatyki i Zarządzania, Politechnika Wroclawska, 2003.*