|  |  |  |  |
| --- | --- | --- | --- |
| Logotyp  C:\Users\Jerry\Documents\doseLOGO.png | Nazwa organizacji  Politechnika Gdańska | | |
| Symbol projektu  DOSE | Nazwa projektu  Document Oriented System Engineering | | |
| Symbol zadania  IML | Nazwa zadania  Model obiektowy semantyki języka IML | | |
| Typ dokumentu  Źródło analizy | Tytuł dokumentu  Język IML – krótka charakterystyka | | |
| Data utworzenia dokumentu  2014-06-29 | Numer wersji  1.0.1 | Edycja | Data ostatniej aktualizacji  2014-06-29 |
| Osoba odpowiedzialna  Jarosław Kuchta | Autor tej wersji dokumentu  Jarosław Kuchta | | Status  Ukończony |

Spis treści

[1. Geneza języka IML 2](#_Toc391803244)

[1.1. Diagramy aktywności UML i wykresy operacyjne 2](#_Toc391803245)

[1.2. Diagramy przepływu danych (DFD) 3](#_Toc391803246)

[1.3. Diagramy interakcji UML 4](#_Toc391803247)

[1.4. Diagramy Nassi-Shneidermana (N-S) 5](#_Toc391803248)

[1.5. Diagramy przejść stanów 5](#_Toc391803249)

[2. Cechy charakterystyczne języka IML 5](#_Toc391803250)

[2.1. Dualizm tekstowo-graficzny 6](#_Toc391803251)

[2.2. Podobieństwo reprezentacji graficznej i tekstowej 6](#_Toc391803252)

[2.3. Minimalizacja zbioru symboli graficznych 7](#_Toc391803253)

[2.4. Elastyczność składni tekstowej 7](#_Toc391803254)

[2.5. Oddzielenie pojęcia nazwy od identyfikatora 8](#_Toc391803255)

[2.6. Stosowanie tekstu wielojęzycznego 8](#_Toc391803256)

[2.7. Wyróżnianie słów kluczowych 9](#_Toc391803257)

[2.8. Wykorzystanie stereotypów do precyzowania semantyki 9](#_Toc391803258)

[Bibliografia 10](#_Toc391803259)

# Geneza języka IML

Język modelowania i implementacji IML zapewnia środki do prezentacji projektu w sposób graficzny i w sposób tekstowy. Za podstawę definicji notacji graficznej przyjęto zunifikowany język modelowania UML [1]. Przy opracowywaniu notacji tekstowej przyjęto wiodące rozwiązania współczesnych języków obiektowych, takich jak Delphi, C#, C++, Java i Visual Basic.

Dla zapewnienia spójności notacji graficznej i tekstowej konieczne stało się wprowadzenie pewnych modyfikacji w języku IML w stosunku do UML. Za podstawowy środek graficznej prezentacji funkcjonalności przyjęto znane z UML diagramy aktywności, do których wprowadzono elementy stosowane na wykresach operacyjnych oraz elementy przepływu danych wzorowane na diagramach DFD (stosowanych np. w metodzie OMT [2]) i na diagramach interakcji UML. Notację przepływu danych znacznie wzbogacono dla wyrażenia szczegółów implementacyjnych. Do diagramów aktywności wprowadzono elementy sterowania strukturalnego wzorowane na diagramach Nassi-Shneidermana [3]. Połączono diagramy aktywności z diagramami interakcji i diagramami przejść stanów tak, że stanowią one bezpośrednie rozszerzenie diagramów aktywności.

## Diagramy aktywności UML i wykresy operacyjne

*Diagramy aktywności* UML przypominają stosowane od lat 1960 wykresy operacyjne (ang. flowchart). W języku IML wprowadzono do definicji diagramu aktywności pewne modyfikacje w zakresie symboliki, z których najważniejsze to:

* zastosowanie sześciokąta jako symbolu warunku,
* zastosowanie prostokąta zaokrąglonego jako symbolu aktywności i prostokąta zwykłego jako symbolu akcji,
* zastosowanie wektora przerywanego jako wektora przepływu sterowania i wektora ciągłego jako wektora przepływu danych.



Rys. 1. Oznaczenie rozgałęzienia sterowania na diagramie aktywności: a) w języku UML, b) w języku IML

*Sześciokąt* jako *symbol warunku* jest bardziej pojemnym symbolem od rombu w tym sensie, że można w nim zmieścić więcej tekstu. Jest to ważne przy praktycznym wykorzystaniu tego symbolu w implementacji. Przy zastosowaniu rombu do rozgałęzienia sterowania (jak w UML), przy każdym wektorze odchodzącym od tego symbolu umieszcza się wyrażenie warunku. Jest to wygodne przy modelowaniu, tymczasem przy imple­mentacji oblicza się jedno wyrażenie dla warunku i w zależności od wartości tego wyrażenia podejmuje się decyzje co do dalszego postępowania (p. rys. 1).

Drugą modyfikacją jest wprowadzenie do diagramu aktywności IML oprócz *symbolu aktywności* (prostokąta zaokrąglonego– jak w diagramach UML) również *symbolu akcji* (zwykłego prostokąta – jak na wykresach operacyjnych). *Aktywność* jest działaniem długotrwałym, podejmowanym często cyklicznie. *Akcja* z kolei jest działaniem krótkotrwałym, niepodzielnym w skali czasowej diagramu (p. rys. 2).



Rys. 2. Różnice w stosowaniu prostokąta zaokrąglonego i prostego:   
a) symbol aktywności, b) symbol akcji

Ponieważ diagram interakcji stanowi proste rozszerzenie diagramów aktywności, więc na tym ostatnim mogą występować dwa rodzaje przepływu: przepływ danych i przepływ sterowania. Dlatego wprowadzono dwa różne symbole przepływu: *wektor przerywany* dla *przepływu sterowania* i *wektor ciągły* dla *przepływu danych* . Diagram aktywności z wektorami przepływu sterowania przedstawiono na rys. 3. Dla porównania podano też prezentację tego samego algorytmu w notacji UML i w notacji wykresów operacyjnych.

|  |  |  |
| --- | --- | --- |
|  |  |  |

Rys. 3. Diagram aktywności z wektorami przepływu sterowania: a) w notacji IML, b) w notacji UML, c) w notacji stosowanej na wykresach operacyjnych

## Diagramy przepływu danych (DFD)

W definicji języka UML zabrakło *diagramów przepływu danych* (ang. Data Flow Diagrams – DFD) opisujących funkcjonalność poprzez przetwarzanie danych między procesami. Zostały one zastąpione przez diagramy interakcji, które jednak nie zawierają symboli procesów. Z drugiej strony w UML są diagramy aktywności, które prezentują procesy. W IML wprowadzono elementy przepływu danych do diagramów aktywności w ten sposób, że:

* Węzłami przepływu mogą być *obiekty* i *operacje*. Symbolem obiektu jest *ikonid* (zespół złożony z *ikony* i *pola tekstowego* identyfikującego obiekt), symbolem operacji jest prostokąt zaokrąglony (jeśli operacja jest realizowana w formie *aktywności*) lub prostokąt zwykły (jeśli operacja jest realizowana w formie *akcji*).
* Operacje mogą nie tylko przetwarzać dane, ale również być źródłem i miejscem przeznaczenia danych.
* Węzły przepływu danych są połączone *wektorami przepływu danych* w formie wektora ciągłego zakończonego strzałką prostą . Wektory mogą mieć trzy etykiety: źródłową, główną i docelową. *Etykieta wektora* przepływu danych jest *polem tekstowym*, które opisuje właściwości lub operacje obiektów źródłowych i docelowych wektora będące źródłem lub przeznaczeniem danych, albo parametry operacji źródłowych i docelowych wektora.
* Diagram umożliwia pokazanie *wymiany danych* między dwoma obiektami i modyfikację danych przez parametr modyfikowalny operacji. Służy do tego *dwukierunkowy wektor przepływu danych*.
* Zamiast ikonidu obiektu może być stosowane pole tekstowe zawierające *wyrażenie*.
* Diagram umożliwia wybór różnych źródeł danych i różnych miejsc przeznaczenia danych przez umieszczenie *symbolu warunku* na drodze przepływu danych.

Przykład pokazano na rys. 4. Dla porównania podano też prezentację tego samego diagramu w notacji DFD.

|  |  |  |  |
| --- | --- | --- | --- |
| *a)* |  | *b)* |  |

Rys. 4. Diagram przepływu danych: a) w notacji IML, b) odpowiadający mu diagram DFD

## Diagramy interakcji UML

W UML przepływ danych wyraża się w formie *diagramów interakcji*. *Interakcja* reprezentuje wymianę komunikatów między obiektami współpracującymi ze sobą w ramach *kolaboracji*. Kolaborację wyraża się na *diagramie kolaboracji* w formie sieci powiązań między obiektami. Przekazywane wzdłuż tych powiązań komunikaty tworzą interakcję. W IML diagram interakcji jest tożsamy z diagramem aktywności, na którym stosuje się wektory przepływu danych. Przekazywanie komunikatów uogólniono do przesyłania danych. Drogę przesyłania danych między dwoma obiektami wyraża *symbol magistrali danych* (strzałka blokowa). Przykład diagramu interakcji w notacji IML i UML pokazano rys. 5.

|  |  |
| --- | --- |
| *a)* |  |
| *b)* |  |

Rys. 5. Diagram aktywności z magistralami danych: a) w notacji IML,   
b) odpowiadający mu diagram interakcji w notacji UML

## Diagramy Nassi-Shneidermana (N-S)

Do języka IML zaadaptowano *strukturalne diagramy sterowania* Nassi-Shneidermana (N-S). Diagramy te stosowano w latach 1970 dla rozwiązania problemu strukturalizacji algorytmu, czyli dopasowania notacji graficznej do *strukturalnych języków programowania*. Aby zapewnić spójność notacji diagramów N-S z symboliką stosowaną w diagramach aktywności wprowadzono pojęcie *graficznej struktury sterowania* konstruowanej z podstawowych symboli: bloku wykonawczego  (symbolu akcji), bloku powtarzania  (symbolu aktywności), symbolu warunku  oraz symbolu ramki grupującej . Graficzną strukturę sterowania tworzy się poprzez nakładanie na jednego symbolu na krawędź górną albo dolną drugiego symbolu. Graficzne struktury sterowania wchodzą jako węzły do diagramów aktywności. Przykład graficznej struktury sterowania pokazano na rys. 6.

|  |  |  |  |
| --- | --- | --- | --- |
| *a)* |  | *b)* |  |

Rys. 6. Algorytm z rys. 3. w notacji strukturalnej:   
a) graficznej struktury sterowania IML, b) diagramu N-S

## Diagramy przejść stanów

Graficzne struktury sterowania mogą być użyte do prezentacji złożonych *stanów* na *diagramie przejść stanów*. Na diagramach UML symbolem stanu jest zaokrąglony prostokąt oznaczający również aktywność. W języku IML rozdzielono symbol stanu i symbol aktywności przyjmując *elipsę* za *symbol stanu*. *Prostokąt zaokrąglony* symbolizuje na diagramach przejść stanów *aktywność*, czyli operację wykonywaną tak długo, jak długo obiekt znajduje się w określonym stanie. Z kolei *prostokąt zwykły* symbolizuje na diagramie przejść stanów *akcję wejściową* lub *wyjściową* stanu albo *akcję przejściową* (wykonywaną w czasie *przejścia stanów*). Ponieważ przejście stanów może być uważane za specyficzny rodzaj przepływu sterowania, więc jako *symbol przejścia stanów* używa się *wektora przepływu sterowania*. *Symbol warunku* może być również użyty w połączeniu z wektorem przejścia stanów do oznaczenia *warunku strzegącego* przejścia. Przykład diagramu stanów IML pokazano na rys. 7. Dla porównania pokazano diagram przejść stanów w notacji UML.

|  |  |  |  |
| --- | --- | --- | --- |
| *a)* |  | *b)* |  |

Rys. 7. Diagram stanów: a) w notacji IML, b) w notacji UML

# Cechy charakterystyczne języka IML

W języku IML zastosowano kilka charakterystycznych rozwiązań, takich jak:

* dualizm tekstowo-graficzny,
* podobieństwo reprezentacji graficznej i tekstowej,
* minimalizacja zbioru symboli graficznych,
* elastyczność składni tekstowej,
* oddzielenie pojęcia nazwy od identyfikatora,
* stosowanie tekstu wielojęzycznego,
* wyróżnianie słów kluczowych,
* wykorzystanie stereotypów do precyzowania semantyki.

## Dualizm tekstowo-graficzny

Reprezentacja graficzna projektu nadaje się do projektowania na wysokim poziomie abstrakcji. Diagramy ułatwiają pracę koncepcyjną, umożliwiają lepsze zrozumienie powiązań między elementami projektu. Z drugiej strony reprezentacja tekstowa projektu bardziej nadaje się do wyrażenia dużej liczby szczegółów implementacyjnych. W przypadku prostych algorytmów szybciej pisze się tekst programu niż rysuje diagram przepływu. Dlatego w metodzie IMAC przyjęto w warstwie prezentacji języka IML zasadę *dualizmu tekstowo-graficznego*. Zasada ta oznacza, że *każdy element semantyczny może mieć reprezentację tekstową albo graficzną, a wybór reprezentacji zależy od projektanta* (rys. 8). W połączeniu z regułą podobieństwa reprezentacji graficznej i tekstowej (p. poniżej) zapewnia to możliwość automatycznej (lub półautomatycznej) zmiany reprezentacji.

|  |  |
| --- | --- |
|  | for i ← 1 to A.Row.Count do {  for j ← 1 to B.Col.Count do {  C[i, j] ← sum **{** from k ←1 **to** A.Col.Count  **{** A[i,k] ∙ B[k,j] **}**  **}**  }  } |

Rys. 8. Reprezentacja graficzna i odpowiadająca jej reprezentacja tekstowa dla   
algorytmu mnożenia macierzy

## Podobieństwo reprezentacji graficznej i tekstowej

Z zasady dualizmu tekstowo-graficznego wynika *reguła podobieństwa* reprezentacji graficznej i reprezentacji tekstowej. Ta nieformalna reguła stanowi, że *reprezentacja graficzna i reprezentacja tekstowa elementu są do siebie wizualnie „podobne”.* Dlatego np. za *operator przypisania* w IML przyjęto strzałkę. Symbol strzałki jest „podobny” do wektora przepływu danych, a oba symbole reprezentują te same operacje. W celu jeszcze lepszego dopasowania składni tekstowej i graficznej zdefiniowano trzy symbole przypisania: w lewo ←, w prawo → oraz symbol wymiany ↔. Dwa pierwsze mogą być łączone w symbol operacji złożonej (np. „addTo”) z symbolami operacji binarnych (takimi jak ‘+’) poprzez umieszczenie symbolu operacji binarnej w indeksie górnym bezpośrednio przy końcu docelowym strzałki. Rozpoznaje się też symbol złożony, w którym symbol operacji binarnej jest umieszczony w indeksie górnym pomiędzy strzałką a poziomą kreską ją przedłużającą (rys. 9).

|  |  |
| --- | --- |
|  | A ← B |
|  | A +← B |
|  | A ←+⎯ B |

Rys. 9. Przykłady podobieństwa reprezentacji graficznej i tekstowej

## Minimalizacja zbioru symboli graficznych

Pamiętanie znaczenia wielu różnorodnych symboli graficznych byłoby w praktyce zbyt trudne. Dlatego zdecydowano się na ograniczenie ich liczby do kilku prostych kształtów (prostokąt zwykły i zaokrąglony, sześciokąt, romb, elipsa, koło) i rozróżnianie ich poprzez styl linii (ciągły, przerywany). Podobnie jak w składni tekstowej ograniczony zbiór liter, cyfr i symboli tworzy słowa kluczowe, nazwy, liczby i operatory, tak w składni graficznej ograniczony zbiór symboli graficznych (rys. 10) tworzy konstrukcje językowe, których znaczenie jest rozpoznawane poprzez ich wzajemne położenie i relacje między nimi.



Rys. 10. Ograniczony zbiór symboli graficznych języka IML

## Elastyczność składni tekstowej

Za podstawę do zdefiniowania składni tekstowej języka IML przyjęto cztery współczesne języki obiektowe: Delphi (Object Pascal), C++, C#, Java, Visual Basic. Język Delphi umieszczono tu na pierwszej pozycji, bowiem ma jedną niewątpliwą przewagę nad pozostałymi wymienionymi tu językami – jest dużo bardziej zrozumiały. Dotyczy to zarówno operatorów, jak i deklaracji. Dlatego w języku IML wprowadzono operatory xor i mod – zrozumiałe dla każdego informatyka w odróżnieniu od symboli ^ i %, które w różnych językach mają różne znaczenie. Deklaracje operacji rozpoczynają się w IML słowami kluczowymi np. procedure, function, constructor, destructor, co umożliwia szybkie określenie semantyki definicji po pierwszym słowie kluczowym. Aby jednak nie ograniczać stosowania języka IML do grupy zwolenników Delphi, wprowadzono zasadę *elastyczności* *składni*. Wyraża się ona następującymi regułami:

* Słowa kluczowe nie występujące na początku deklaracji ani jako separatory klauzul są nieobowiązkowe. Dla przykładu można napisać zarówno

if ok then DoSomething

jak i

if ok { DoSomething }

* Wiele operatorów ma więcej niż jeden symbol. Stosuje się zarówno symbole znane z wymienionych języków programowania, jak również symbole znane z matematyki (przy wykorzystaniu standardu Unicode). Dla przykładu mnożenie można zapisać jako „a\*b” albo jako „a∙b”.
* Istnieją różne słowa kluczowe o tej samej semantyce (np. „this” i „self”).
* Nazwy mogą mieć skróty i aliasy, co umożliwia skrócenie zapisu i stosowanie synonimów. Przykładem skrótu dla nazwy „Formularz Złożenia Zamówienia” może być „FZZ”.

## Oddzielenie pojęcia nazwy od identyfikatora

W języku IML rozróżnia się pojęcie nazwy i identyfikatora. O ile nie precyzuje się inaczej, to pod pojęciem nazwy rozumie się nazwę prezentacyjną, a pod pojęciem identyfikatora – identyfikator kanoniczny tworzony na podstawie tej nazwy.

*Nazwa prezentacyjna* służy do reprezentowania elementu projektu na diagramie i w tekście języka IML. Ponieważ jednak metoda IMAC przewiduje generowanie kodu dla środowiska wykonania w różnych językach programowania, które niekoniecznie dopuszczają stosowanie innych liter niż łacińskie i innych cyfr niż dziesiętne, więc elementy identyfikowane są nie po samych nazwach, ale po identyfikatorach tworzonych na podstawie nazw. Są to tzw. *identyfikatory kanoniczne*. Identyfikator kanoniczny rozpoczyna się od litery i zawiera tylko litery i cyfry. Pod pojęciem litery rozumie się tu tylko litery łacińskie (od ‘A’ do ‘Z’ i od ‘a’ do ‘z’), a pod pojęciem cyfry tylko cyfry dziesiętne (od ‘0’ do ‘9’). Znak podkreślenia jest traktowany jak litera. Litery małe i wielkie mogą, lecz nie muszą być rozróżnialne. Przykład – tab. 1

Tab. 1. Przykłady nazw i identyfikatorów

|  |  |
| --- | --- |
| **Nazwa prezentacyjna** | **Identyfikator kanoniczny** |
| błąd | blad |
| π | pi |
| długa nazwa | dluga\_nazwa |

Oprócz identyfikatorów kanonicznych elementy języka IML mają jeszcze dwa inne identyfikatory: *identyfikator unikatowy* (ID) służący do powiązania obiektowej reprezentacji języka z bazą danych i *identyfikator referencyjny* (refID) służący do odwoływania się do elementów niezależnie od ich nazwy prezentacyjnej.

## Stosowanie tekstu wielojęzycznego

W świecie współczesnym nie można ograniczać możliwości pracy programisty do języka angielskiego. Dlatego podobnie, jak w języku Java i C# możliwe jest stosowanie nazw zawierających litery Unicode. Ponadto zauważono, że o ile analityk posługuje się często nazwami pochodzącymi ze swojego rodzimego języka, o tyle programista posługuje się najczęściej nazwami angielskimi. Jest to wymuszane przez środowisko implementacji (biblioteki wykonawcze), dla którego produkowany jest kod. Dlatego w IML wprowadzono pojęcie *tekstu wielojęzycznego*. Pod tym pojęciem rozumie się kolekcję elementów tekstowych, z których każdy składa się z pojedynczego łańcucha znakowego i trzyliterowego identyfikatora języka, w którym ten tekst został zapisany (zgodne z normą ISO 639). Wprowadzane w czasie analizy nazwy, opisy i komentarze są oznaczane identyfikatorem języka narodowego. Przy definiowaniu elementów projektant powinien również wprowadzać nazwy w języku angielskim. Wytworzenie kodu lub dokumentacji w języku angielskim lub narodowym wymaga później tylko wyboru odpowiedniego języka.

## Wyróżnianie słów kluczowych

W wielu językach programowania występują słowa kluczowe, czyli ściśle określone słowa, które nie mogą być używane jako nazwy, lecz służą do odpowiedniego rozpoznawania składni tekstu programu. Najczęściej rozpoznawane słowa kluczowe, to „if”, „else”, „for”, „true”, „false”. Jednak różne języki programowania mają różne zbiory słów kluczowych. Aby umożliwić wykorzystanie gotowych komponentów napisanych w różnych językach programowania i nie dopuścić do konfliktu nazw ze słowami kluczowymi, w języku IML każde słowo, które ma być w określonym kontekście użyte jako słowo kluczowe, musi być w specjalny sposób oznaczone. Do oznaczania słów kluczowych zastosowano wytłuszczenie. Oznacza to, że tekst źródłowy w języku IML musi być tekstem formatowanym, z zapisanymi w nim *atrybutami typograficznymi*. Przykład tekstu formatowanego z wyróżnionymi słowami kluczowymi przedstawia rys. 11.



Rys. 11. Przykład tekstu w języku IML z wyróżnionymi słowami kluczowymi

## Wykorzystanie stereotypów do precyzowania semantyki

Stereotyp jest prezentowany jako słowo kluczowe występujące na początku instrukcji lub deklaracji języka IML albo poprzedzające nazwę elementu. Może być to pojedyncze słowo kluczowe lub rozpoznawalna sekwencja słów. W przykładzie z rys. 11 stereotypem instrukcji jest „for each”.

W języku IML stosuje się ograniczony zbiór elementów graficznych do wyrażenia bogatej semantyki implementacyjnej. W wielu przypadkach semantykę elementu graficznego trzeba sprecyzować przez rozpoznanie stereotypu tekstu umieszczonego wewnątrz tego elementu. Tak jest np. w przypadku sześciokąta, która to figura reprezentuje na diagramach warunek. Umieszczenie sześciokąta na górnej krawędzi prostokąta zaokrąglonego jest tłumaczone na instrukcję iteracji. Stereotyp instrukcji jest rozpoznawany po sekwencji słów kluczowych rozpoczynających tekst umieszczony wewnątrz sześciokąta (rys. 12).



Rys. 12. Rozróżnienie semantyki elementu graficznego przez stereotyp: a) „*for each*”, b) „*while*”

# Bibliografia

|  |  |
| --- | --- |
| [1] | G. Booch, J. Rumbaugh and I. Jacobson, The Unified Modeling Language User Guide (2nd ed.), Addison-Wesley, 2005. |
| [2] | J. Rumbaugh, Object-Oriented Modeling and Design, Prentice-Hall, 1991. |
| [3] | I. R. Nassi i B. Shneiderman, „Flowchart techniques for structured programming,” *ACM SIGPLAN Notices,* pp. 12-26, August 1973. |
| [4] | J. Kuchta, „Integracyjna metoda wytwarzania aplikacji obiektowych w środowisku graficznym z uwzględnieniem wymagań jakościowych,” Wydz. ETI PG, Gdańsk, 2008. |