Andrzej Sobczak

11. Nowe języki opisu wymagań dla informatycznych systemów zarządzania

Streszczenie: W rozdziale przedstawiono języki opisu wymagań dotyczących systemów informatycznych zarządzania, nad którymi podjęto prace badawcze w przeciągu ostatnich kilku lat. W porównaniu ze starszymi rozwiązaniami, następuje tutaj zmiana paradygmatu. Pozwalają one nie tylko dokumentować wymagania, ale również wskazywać kontekst, w jakim będzie funkcjonował budowany system oraz próbują odwzorować cele, dla których dane wymaganie zostało wprowadzone. Dotyczy to zwłaszcza wymagań niefunkcjonalnych dla systemów, takich jak: bezpieczeństwo, wydajność, ergonomiczność.

Słowa kluczowe: inżynieria wymagań, ontologia, agent, cel.

Wprowadzenie

W pracy [5] można znaleźć następującą klasyfikację sposobów opisu wymagań:

- Język tekstowy – można wykorzystywać język naturalny lub jego postać formalną. Bez wątpienia język naturalny ma lepszą siłę przekonywania i większą swobodę ekspresji. Jest on również bliższy przeciętnemu użytkownikowi systemów informatycznych. Z drugiej jednak strony, wymagania wyspecyfikowane w języku naturalnym mogą być różnie interpretowane przez różnych interesariuszy. W tym zakresie lepsze są więc języki mające precyzyjnie określoną składnię i bardziej restrykcyjną semantykę.
- Język graficzny – generalnie panuje zgoda, że graficzna reprezentacja skomplikowanego materiału jest znacznie bardziej zrozumiała niż jej tekstowy odpowiednik. Wynika to m.in. z faktu że:
  - Grafika jest dwuwymiarowa, tekst jest jednowymiarowy. Grafika zapewnia więc dodatkowy stopień swobody prezentacji.
  - Grafika jest bardziej przydatna do pokazywania hierarchicznej struktury skomplikowanych systemów i bardziej naturalna w opisywaniu paralelizmu (np. kroków w algorytmie).
  - Osoba czytająca grafikę może robić to selektywnie, w zależności od żądanego poziomu szczegółowości. Czytając tekst, trzeba robić to liniowo.
  - Liczba pojęć, które może rozsądnie pomieścić ludzka pamięć krótkotrwała jest ograniczona. Osoba czytająca grafikę może zacząć od ogółów i przechodzić do szczegółów po częściowym oswojeniu się z tematem. Jeżeli ktoś czyta tekst, musi zacząć od szczegółów, abstrahując pojęcia zasadnicze później.

  Należy jednak zauważyć, że język graficzny ze zbyt dużą ilością symboli niekoniecznie jest zrozumiały dla użytkowników końcowych. Tak więc język oparty na grafice musi składać się ze stosunkowo niewielkiej ilości łatwo zrozumiałych symboli.
- Język hybrydowy – często niepraktyczne jest określanie specyfikacji wymagań tylko w jednym (graficznym lub tekstowym) języku. Chociaż graficzne języki lepiej nadają się niż języki tekstowe do prezentowania przeglądu

skomplikowanej sytuacji, to języki tekstowe są uznawane za lepsze do szczegółowego opisu.

Na bazie powyższej klasyfikacji powstało wiele języków służących do opisu wymagań – takich jak PSL, SADT, EDDA, SAMM, HOS, RSL. Są to rozwiązania starszej generacji (pochodzą z lat 80. XX w.) i obecnie nie są już praktycznie rozwijane. Od kilku lat podejmowane są natomiast prace badawcze, mające na celu opracowanie nowej generacji języków do opisu wymagań, które pozwalają na: wskazanie kontekstu, w jakim będzie funkcjonował budowany system oraz próbują odwzorować cele, dla których zostało wprowadzone dane wymaganie (dotyczy to w szczególności wymagań niefunkcjonalnych).

11.1. Przegląd nieklasycznych języków opisu wymagań

11.1.1. Goal-oriented Requirement Language (GRL)

Język GRL [4] jest językiem opisu wymagań do wspierania modelowania ukierunkowanego na cele (*goals*) i agentów. Zapewnia on efektywną metodę opisu wymagań niefunkcjonalnych. Język ten operuje trzema głównymi kategoriami pojęć: elementy zamierzone, powiązania oraz aktorzy.

Tzw. elementami zamierzonymi w GRL są: cel, zadanie, „miękki cel" (*softgoal*) i zasób. Noszą one nazwę „zamierzonych", ponieważ są stosowane jako składowe modelu, umożliwiające udzielanie odpowiedzi na pytania, dlaczego poszczególne zachowania i struktury informacyjne zostały wybrane podczas specyfikacji wymagań, jakie alternatywy były rozważane, jakie kryteria zostały zastosowane przy rozpatrywaniu alternatywnych możliwości, a także jakie były powody wyboru tej a nie innej alternatywy.

Specyfikacja wyrażona w języku GRL może się składać albo z globalnego modelu celów, albo z serii modeli celów rozproszonych wśród kilku aktorów. Jeżeli model celów zawiera więcej niż jednego aktora, wtedy można wyrazić i rozpatrywać zamierzone powiązania pomiędzy aktorami.

Cel jest okolicznością lub stanem rzeczy w świecie, który chcieliby osiągnąć interesariusze. Na ogół nie jest określone, jak ten cel ma być osiągnięty, dzięki czemu istnieje możliwość rozważania alternatywnych rozwiązań. Cel może być albo biznesowy albo systemowy. Cele biznesowe dotyczą stanu zagadnień biznesowych, jaki interesariusz pragnie osiągnąć. Cele systemowe dotyczą tego, co powinien realizować system informatyczny. Zazwyczaj jest to opis funkcjonalnych wymagań budowanego systemu.

Zadanie specyfikuje konkretny sposób wykonania określonej czynności. Gdy zadanie jest specyfikowane jako podkomponent zadania wyższego rzędu, ogranicza to zadanie wyższego rzędu do określonego toku postępowania. Zadania można również uważać za rozwiązania zaspokajające „miękkie cele" (*softgoals*) lub prowadzące do osiągnięcia określonego celu.

„Miękki cel" jest okolicznością lub stanem, który chciałby osiągnąć aktor, ale w przeciwieństwie do „twardego" celu – nie istnieją wyraźne kryteria oceny, kiedy stan taki jest osiągnięty. Projektant musi sam osądzić, czy dany stan rzeczywiście stanowi osiągnięcie podanego „miękkiego celu". „Miękkie cele" służą do reprezentowania wymagań niefunkcjonalnych (takich jak: wydajność, bezpieczeństwo, możliwość powtórnego wykorzystania, interoperacyjność, czas wprowadzenia na rynek i koszt). Należy się nimi zająć możliwie w jak najwcześniejszej fazie „cyklu życia"

systemu i powinny one znaleźć swoje odzwierciedlenie w jego architekturze zanim zostanie podjęta decyzja co do określonej implementacji.

Zasób jest jednostką (fizyczną lub informatyczną), a główną troską jest to, czy jest on dostępny.

Do zamierzonych powiązań języka GRL należą powiązania: środków-celów (*means-ends*), rozkładu (*decomposition*), przyczynku (*contribution*), korelacji (*correlation*) i zależności (*dependency*). Powiązania środków-celów są stosowane do opisu jak w rzeczywistości osiągane są zidentyfikowane cele. Każde zadanie, połączone z celem powiązaniem środków-celów, jest jedną z możliwych dróg prowadzących do osiągnięcia określonego celu. Rozkłady określają, jakie inne podelementy muszą być osiągnięte lub dostępne, żeby dane zadanie było wykonane. Powiązania przyczynku opisują, jak „miękkie cele", zadania i pozostałe powiązania przyczyniają się do innych.

Specyfikacja wymagań z zastosowaniem języka GRL różni się od szczegółowej specyfikacji. Tutaj twórca specyfikacji zajmuje się głównie eksponowaniem „dlaczego" zostały dokonane określone wybory zachowań i struktury i dlaczego zostały wprowadzone pewne ograniczenia. Twórca specyfikacji nie interesuje się jeszcze „operacyjnymi" szczegółami wymagań procesów lub systemu (ani wzajemnym oddziaływaniem komponentów). Pomijanie tego rodzaju szczegółów we wczesnej fazie analizy (i projektowania) umożliwia szersze (zwane czasem strategicznym) spojrzenie na modelowanie istniejącego lub przyszłego systemu oprogramowania i jego otoczenia. Modelowanie i odpowiadanie na pytania „dlaczego" prowadzi do rozważań na temat możliwości, które interesariusze starają się uzyskać i niedoskonałości, których próbują uniknąć w swoim środowisku przez wykorzystanie zdolności systemu lub innych interesariuszy, a także starając się polegać na możliwościach lub tworząc możliwości i wprowadzając ograniczenia na sposoby, w jakie te możliwości powinny zostać zrealizowane.

11.1.2. KAOS

KAOS jest językiem opisu wymagań zorientowanym na cele [3]. Oznacza to, że po zidentyfikowaniu kilku wstępnych celów dla przyszłego systemu, umożliwia on identyfikację wymogów, obiektów, agentów i czynności systemu. KAOS zapewnia bogatą ontologię do uchwycenia i modelowania wymagań. Jej meta-poziom jest taksonomią pojęć sterujących identyfikacją wymagań i ich wzajemnymi zależnościami. KAOS zapewnia możliwość zdefiniowania sieci pojęć dla danej dziedziny. Cele są analizowane i lepiej rozumiane przez określenie celów wyższego rzędu, które wyjaśnią, dlaczego ten cel jest pożądany – a są one bardziej dokładnie definiowane przez zredukowanie ich do celów podrzędnych (*subgoals*).

Przez identyfikowanie zależności, jakie cel ma względem innych celów powstaje graf celów (*goal graph*). Grafy celów są semantycznymi sieciami zależności AND/OR/XOR między celami, gdzie cel jest albo redukowany przez koniunkcję celów podrzędnych, albo przez dysjunkcję. Można również śledzić zależności konfliktogenne między celami. Liście grafów celu są wymogami – celami, których nie można dalej redukować i które mogą być przyporządkowane jako obowiązek do poszczególnych agentów. Wymóg jest albo przyporządkowywany agentowi oprogramowania i uznawany za wymaganie, albo agentowi w dziedzinie i uznawany za założenie. Rozróżnienie między wymaganiami i założeniami wyznacza granicę między oprogramowaniem a otoczeniem przyszłego systemu. Intuicyjnie, wymagania są

stwierdzeniami preskryptywnymi, a założenia deskryptywnymi. Wymogi są operacjonalizowane przez czynności wykonywane przez agentów odpowiedzialnych za każdy wymóg w ten sposób, że system realizuje określone cele.

Agenci mogą również monitorować i sterować pewnymi obiektami identyfikowanymi podczas określania celów. Obiekty te mogą być wejściami lub wyjściami czynności. Cele dotyczą obiektów, a obiekty zapewniają realizację wymogu, jeżeli uczestniczą w działaniu operacjonalizującym ten wymóg. Zidentyfikowanie ewentualnych przeszkód na drodze do celu umożliwia przeformułowanie grafu celów i wzmocnienie specyfikacji wymagań przez wybór alternatywnych dróg redukcji lub przydziałów agentów, albo też przez wprowadzenie nowych celów do „obniżenia" przeszkody.

KAOS wprowadza formalny język z operatorami temporalnymi czasu rzeczywistego, który może być zastosowany gdy konieczna jest formalizacja specyfikacji.

11.1.3. i*

Nazwa języka i* (czytaj: eye-star) odnosi się do pojęcia rozproszonej intencjonalności [2]. Opiera się ona na założeniu, że w tej fazie ważne jest zrozumienie i modelowanie środowiska społecznego, w którym ostatecznie będzie funkcjonował system i* umożliwia dokonanie nieformalnego graficznego opisu otoczenia organizacyjnego, które później może być opisane w formalnym języku bardziej nadającym się do automatycznej analizy.

Język i* oferuje trzy kategorie pojęć wyprowadzonych z języków zorientowanych na cele i agentów: aktorzy, zamierzone elementy i powiązania. Aktor jest aktywną całością, która wykonuje czynności, aby osiągnąć swoje cele.

Do zamierzonych elementów w i* należą: cele, miękkie cele, zadania i zasoby. Cel jest warunkiem lub stanem rzeczy, który chcieliby osiągnąć interesariusze. Miękki cel, w typowym przypadku, jest niefunkcjonalnym wymaganiem bez wyraźnie określonych kryteriów mówiących, kiedy jest on osiągnięty. Zadanie specyfikuje określony sposób działania, który daje pożądany efekt.

W i*, udoskonalenia i zależności między zamierzonymi elementami są przedstawiane za pomocą powiązań, które obejmują środki-cele, dekompozycję i powiązania przyczynowe (*contribution links*). Każdy element, połączony z celem za pomocą powiązania środki-cele, jest alternatywnym sposobem osiągania celu. Powiązanie przyczynowe opisuje wpływ, jaki jeden element wywiera na inny. Może on być ujemny (-) lub dodatni (+).

11.1.4. Tropos

Nazwa języka Tropos pochodzi od greckiego τροπος, co oznacza „sposób robienia rzeczy", a także τροπή, co oznacza „kolejność" lub „zmiana" [2].

Tropos proponuje metodologię specyfikacji wymagań, opartą na pojęciach stosowanych do modelowania wczesnych wymagań (*early requirements*). W szczególności propozycja adoptuje podejście zaproponowane przez Eric'a Yu [4]. Tropos jest oparty na dwóch bazowych koncepcjach. Po pierwsze, pojęcie agenta i wszystkie związane z nim pojęcia mentalistyczne (np. cele i plany), są stosowane we wszystkich fazach rozwoju oprogramowania, od wczesnej analizy aż do rzeczywistej implementacji. Po drugie, trzeba zrozumieć otoczenie, w którym będzie pracować

budowany system i rodzaje wzajemnych oddziaływań pomiędzy oprogramowaniem a ludźmi, którzy są agentami.

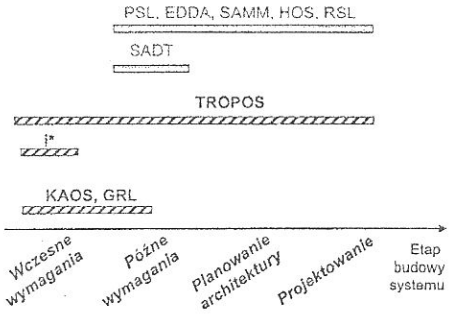
Tzw. Formalny Tropos (*Formal Tropos – FT*) został stworzony w celu uzupełnienia modeli przygotowanych w języku i* o precyzyjny opis ich aspektów dynamicznych [2]. W FT następuje koncentracja nie tylko na samych elementach zamierzonych, ale także na okolicznościach, w jakich one powstają i na warunkach prowadzących do ich spełnienia. W ten sposób na etapie strategicznym wprowadzane są dynamiczne aspekty specyfikacji wymagań, bez konieczności jej operacjonalizacji. Dzięki specyfikacji FT można zadawać następujące pytania: Czy możemy zbudować prawidłowe scenariusze operacyjne na podstawie tego modelu? Czy jest możliwe osiągnięcie celów aktorów? Czy powiązania dekompozycyjne (*decomposition links*) i wcześniejsze ograniczenia (*prior-to constraints*) indukują właściwy porządek czasowy? Czy zależności reprezentują prawidłową synergię oraz synchronizację między aktorami?

Specyfikacja FT opisuje obiekty z dziedziny i relacje między nimi. Opis każdego obiektu jest strukturalizowany w dwóch warstwach. Warstwa zewnętrzna jest podobna do deklaracji klasy i definiuje strukturę wystąpień razem z ich atrybutami. Warstwa wewnętrzna wyraża ograniczenia dotyczące czasu życia obiektów, podane w typowanej liniowej logice temporalnej pierwszego rzędu. Specyfikacja FT jest uzupełniona przez zestaw globalnych własności, wyrażających własności w domenie jako całości.

Podsumowanie

Przedstawione w artykule nieklasyczne języki opisu wymagań w istotny sposób bazują na konceptach i metodach inżynierii wiedzy. Związane jest to z faktem, iż coraz większą uwagę przywiązuje się do podejścia, zgodnie z którym *najistotniejsza jest wiedza ukryta w ludziach i w organizacji; trzeba ją wyartykułować, sformalizować i dopiero na tej podstawie budować system informatyczny* [1].

Jednocześnie analiza zaprezentowanych języków (porównaj: rys. 1) pozwala stwierdzić, że nieklasyczne języki opisu koncentrują się na wczesnym etapie budowy systemu informatycznego.



Rys. 1. Wykorzystanie języków opisu wymagań w poszczególnych etapach budowy systemów informatycznych (linią przerywaną oznaczono języki przedstawione w artykule)
Źródło: opracowanie własne

NON-CLASSICAL LANGUAGES FOR THE REQUIREMENTS DESCRIPTION

Abstract: This chapter reports the results of exploratory research at the area of non-classical languages for requirements representation. Review four languages has been done.

Key words: requirements engineering, ontology, agent, goal.

Literatura

1. Cempel Cz., *Społeczeństwo wiedzy - nowy wymiar kreowania i użytkowania wiedzy*, Serwis eGov.pl, kwiecień 2003
2. Fuxman A., Liu L., Mylopoulos J. Pistore M. Roveri M., Traverso P., *Specifying and Analyzing Early Requirements in Tropos*, Requirements Engineering Journal, 9(2), 2004, s. 132-150
3. Heaven W., Finkelstein A., *A UML Profile to Support Requirements Engineering with KAOS, IEEE Proceedings – Software*, vol. 151, 2004, s. 10-27
4. Liu L., Yu E., *From Requirements to Architectural Design – Using Goals and Scenarios*, Proceedings of the First International Workshop on From Software Requirements to Architectures, Toronto, Ontario, Canada, May 14, 2001
5. Overmyer S., *A Methodology for Constructing User-Oriented Requirements Specifications for Large-Scale Systems Using Electronic Hypermedia*, Requirements Engineering, Springer, Volume 4, Number, May 1998, s. 1-18