

Wizja pracy badawczej w zakresie deklaracyjnych procesów pracy (workflow)

Autor koncepcji:	Kazimierz Subieta, Polsko-Japońska Wyższa Szkoła Technik Komputerowych
Data tej wersji:	15 listopada 2008
Tytuł:	Deklaracyjne obiektowe narzędzie do specyfikacji, implementacji i monitorowania procesów pracy
Kontekst pracy:	Ten dokument jest utworzony jako wstępne stadium dla projektu "Systemy przepływu prac o rozszerzonej funkcjonalności" finansowanego przez Ministerstwa Nauki i Szkolnictwa Wyższego, grant Nr NN5163755534.

1. Wstęp

Obecnie technologia workflow rozwija się głównie za sprawą ciał przemysłowych takich jak WfMC lub OMG (propozycje standardów), rozwiązań ad hoc oferowanych przez dziesiątki firm, oraz środowiska akademickiego proponujące różnorodne pomysły teoretyczne w rodzaju sieci Petriego. Moim zdaniem, obecne koncepcje i technologie prowadzą do problemów, z którymi słabo sobie radzą. Do nich należą:

- Dynamiczna zmiana procesu podczas jego działania;
- Równoległe wykonywanie fragmentów procesów z zapewnieniem odpowiedniej synchronizacji (obecne środki synchronizacyjne są zbyt prymitywne i niedostatecznie uniwersalne);
- Równoległe wykonywanie (pod-) procesów na wielu serwerach;
- Zaniechanie kontynuowania procesu lub jego części w wyniku pewnych rozpoznawanych automatycznie sytuacji;
- Reakcja na wyjątki lub zdarzenia zachodzące w środowisku wykonywania procesów lub jego otoczeniu;
- Zarządzanie zasobami (ludźmi, finansami, czasem, infrastrukturą komputerową, infrastrukturą biurową, aparaturą, pojazdami, itd.) w planie abstrakcyjnym (np. dowolny prawnik) oraz konkretnym (np. Nowak, specjalista w zakresie prawa giełdowego);
- Zintegrowanie modelu obiektowego, metamodelu i języka zapytań bliskich stylistycznie i koncepcyjnie do narzędzi projektowania koncepcyjnego *a la* UML (brak niezgodności impedancji pomiędzy modelem biznesowym i modelem implementacyjnym);
- Refleksja umożliwiająca dynamiczne wygenerowanie nowych (pod-) procesów i automatyczne ich uruchomienie.

Idea polega na tym, aby instancje procesów pracy i podprocesów traktować jako struktury danych (obiekty) z określonymi atrybutami formalnymi umożliwiającymi ich odpytywanie przez język zapytań. Z drugiej strony, takie struktury danych byłyby wyposażone w część

proceduralną, która stanowi istotę procesu pracy, umożliwia komunikację pomiędzy wykonawcą pracy a systemem komputerowym oraz umożliwia integrację aplikacji zewnętrznych. Każdy taki (pod-) proces byłby inherentnie równoległy (tak jak się to dzieje w życiu, gdzie równoległych procesów na budowie jest co najmniej tyle, ilu jest budowniczych). Odchodzimy więc od dobrze osadzonej w językach programowania idei przepływu sterowania (*control flow*), czego manifestacją teoretyczną są sieci Petriego. Konceptyjna zmiana polega na hierarchicznym osadzaniu (pod-) procesów *a la* zagnieżdżona sieć PERT (bez wskazania *explicite* jak ma przepływać sterowanie). Jak wiadomo, sieci PERT ustalają tylko, które czynności muszą się zakończyć, aby można było zacząć daną czynność. W tym sensie ustalają przepływ sterowania, ale na znacznie bardziej ogólnym (i znacznie bardziej logicznym) poziomie, niż to ma miejsce w klasycznych diagramach przepływu sterowania. Ponieważ tak czy inaczej dla dowolnego procesu pracy trzeba będzie ustalić, które czynności mają być wcześniej, a które później, konieczne są dalsze elementy. Do nich należą:

- Warunki początkowe uruchomienia instancji procesu;
- Warunki końcowe umożliwiające stwierdzenie, że dany proces się zakończył;
- Lokalne środowisko (struktury danych) procesu umożliwiające zapis pewnych informacji, np., że pewien podproces został zakończony z takim to a takim wynikiem.

Podstawowym założeniem technicznym tej koncepcji jest oparcie jej o podejście stosowe oraz język. SBQL. Dalsze szczegóły tej koncepcji zostaną opisane w tym opracowaniu.

Opracowanie koncepcji będzie połączone z implementacją na bazie systemu ODRA i języka SBQL. Implementacja będzie tworzona w języku Java.

2. Różne aspekty tej koncepcji

Instancje [pod-] procesów

Proces jest zagnieżdżonym obiektem posiadającym:

1. Unikalny wewnętrzny identyfikator;
2. Nazwę zewnętrzną (biznesową);
3. Flagę ustalającą, że jest to instancja procesu, a nie zwyczajny obiekt;
4. Pewną liczbę atrybutów (podobiektów) widocznych z zewnątrz, czyli publicznych;
5. Pewną liczbę podobiektów niewidocznych z zewnątrz, czyli prywatnych dla tego obiektu (opcjonalnie);
6. Jeden wyróżniony podobiekt zawierający kod (część proceduralną) procesu;
7. Jeden wyróżniony podobiekt zawierający kod warunku początkowego (uruchomienia części proceduralnej);
8. Jeden wyróżniony podobiekt zawierający kod warunku końcowego (zakończenia części proceduralnej);
9. Dowolną liczbę powiązań tego obiektu z innymi obiektami dostępnymi w środowisku procesów pracy;
10. Dowolną liczbę powiązań tego obiektu z obiektami przechowującymi niezmienniki obiektów, czyli klas (opcjonalnie, jeżeli mamy model składu z klasami);
11. Dowolną liczbę procesów (opcjonalnie). Budowa każdego zagnieżdżonego procesu jest identyczna jak wyżej.

Omówienie tych punktów:

(1) Unikalny identyfikator wewnętrzny w sposób unikalny identyfikuje dany obiekt, jak zwykle w obiektowości. Jest nieczytelny i niedrukowalny, nie może być bezpośrednio użyty w kodzie źródłowym zapytania lub programu. W innym kontekście (na stosach) jest nazywany referencją. Referencja jest konieczna w podejściu stosowym, m.in. dla realizacji konstrukcji imperatywnych. Występuje również jako wartość obiektu pointerowego, realizującego powiązania pomiędzy obiektami. Może być implicite użyta jako parametr procedur, funkcji lub metod np. w metodach *call-by-reference* i *strict-call-by-value*.

(2) Nazwa zewnętrzna (biznesowa) jest jak zwykle nazwą tego obiektu wypracowaną w trakcie tworzenia projektu aplikacji, nadaną przez administratora lub innego autoryzowanego użytkownika. Służy do identyfikacji obiektu w kodach zapytań/programów źródłowych, ale (jak zwykle w SBA) nie jest unikalna. Może być wiele procesów posiadających tę samą nazwę. Do rozważenia są synonimy nazw (jak w modelu składu M1) lub nazwy przypisane do ról (jak w modelu M2).

(3) Flaga „jestem instancją procesu” służy do rozróżnienia procesów od innych obiektów będących w tym samym środowisku. Może być zaimplementowana jak wyróżniony atrybut o zastrzeżonej nazwie, która pozwoli interpreterowi na rozpoznanie procesów.

(4) Podobiektu (atrybuty) procesu dostępne z zewnątrz (globalne). Służą do identyfikacji procesu oraz ewentualnie do zmiany jego stanu przez dowolne programy zewnętrzne, np. inne

procesy lub moduły administracyjne. Kilka takich atrybutów powinno być wyróżnione, z zastrzeżonymi nazwami, i ew. niemodyfikowalnych z zewnątrz, w szczególności:

- Stan procesu z wartościami: „nieaktywny”, „oczekujący”, „uruchomiony”, „zakończony”, „zawieszony”, „przerwany”, „przekazany do realizacji na innej maszynie”, itp.;
- Czas utworzenia procesu;
- Czas uruchomienia procesu;
- Czas zakończenia procesu;
- Priorytet procesu;

Prócz tych atrybutów możliwe będą dowolne inne, które zostały przewidziane przy tworzeniu definicji danej instancji procesu (o definicjach będzie dalej), np. ścieżka do serwera realizującego obecnie dany proces przekazany do realizacji na innej maszynie. Atrybuty te mogą przechowywać dowolne informacje o zasobach konkretnych lub abstrakcyjnych przydzielonych do danego procesu, przy czym dla procesów uruchomionych i zakończonych pamiętane są zużyte zasoby konkretne, ipt. Atrybuty mogą oczywiście posiadać dowolną licznosc i złożoność: mogą być złożone, opcjonalne, powtarzalne, itd.

(5) Podobiekty (atrybuty) procesu niewidoczne z zewnątrz (lokalne). Założenia podobne do poprzedniego, z tym zastrzeżeniem, że atrybuty te nie mogą być użyte w dowolnych zapytaniach z zewnątrz danej instancji procesu. Natomiast mogą być użyte w dowolnych zapytaniach z wnętrza procesu, czyli w warunkach początkowych i końcowych oraz kodzie procesu. Zarówno podobiekty globalne jak i lokalne będą podstawowym mechanizmem przekazania sterowania z jednego procesu do innego procesu. Jeżeli mamy w tym samym środowisku procesy A i B z założeniem, że B może rozpocząć się dopiero po zakończeniu A, to proces A przy zakończeniu ustawia sobie stan na „zakończony”, natomiast proces B w warunku początkowym ma predykat: $(A.stanProcesu) = „zakończony”$. Oczywiście, nasz model dopuszcza mrowie innych sposobów przekazania sterowania oraz zrównoleglenia działania procesów.

(6) Podobiekt zawierający kod (część proceduralną) procesu jest opcjonalny, w szczególności nie musi wystąpić dla procesów złożonych. Przyjmując, że językiem programowania będzie SBQL, kod ten jest dowolnym ciągiem instrukcji imperatywnych SBQL. Kod ten może być także skompilowanym programem Java lub innego języka programowania, którego kody dają się odpalać dynamicznie. W zasadzie, kod ten może zawierać dowolne instrukcje imperatywne i przyjmujemy roboczo, że jest on jednowątkowy, nie ma elementów rozszczepiających (split) i synchronizacyjnych (join). Wielo-wątkowość można realizować poprzez wiele równoległych procesów. Kilka instrukcji może mieć znaczenie dla procesów pracy, w szczególności:

- Wysłanie maila z dynamicznie tworzoną treścią do określonego adresata;
- Wygenerowanie dokumentu HTML, XML, TXT, PDF lub innego;
- Przesłanie dowolnego dokumentu do określonego miejsca;
- Odpalenie aplikacji zewnętrznej na określonym komputerze z określonymi parametrami i plikiem, w szczególności odpalenie Web Services;
- Zarządzanie czasem, w postaci instrukcji synchronizujących zegary wszystkich komputerów będących pod zarządem naszej aplikacji workflow, ustawiania budzików i reakcja na budziki, itp.

- Współpraca z określoną aplikacją instalowaną po stronie klienta, np. aplikacją ustalającą w sposób graficzny kolejkę zadań do wykonania przez danego wykonawcę (wstawianie zadania do kolejki, odpytywanie kolejki, itp.)
- Współpraca z monitorem przydziału zasobów do realizacji procesów (jeżeli taki monitor powstanie, co jest odrębnym tematem badawczo-implementacyjnym).
- Przekazanie danego (pod-) procesu do realizacji na innej określonej maszynie.

(7) Warunek początkowy (uruchomienia części proceduralnej) jest ewaluowany przez interpreter w określonym przez projektanta cyklu. Oczywiście, im cykl jest dłuższy, tym komputer ma mniej pracy. Cykl badania tego warunku może być np. co sekundę, dla bardzo szybkich procesów wymagających prawie-rzeczywistego czasu reakcji, co minutę, co 10 minut, co godzinę, codziennie, co tydzień, itd. Cykl zależy od skali czasowej w której zachodzi dany biznes. Warunek ma postać dowolnego zapytania (oczywiście SBQL) wartościowanego do true/false, poza tym zapytania może odwoływać się do dowolnych elementów całego środowiska danej aplikacji workflow, w szczególności, do stanu innych (pod-) procesów, lokalnych i globalnych atrybutów, zegara, budzików, wszelkich zasobów abstrakcyjnych i konkretnych, itd. Dopóki warunek zwraca *false*, nic się nie dzieje. W momencie, w którym warunek przyjmuje wartość *true*, proces jest uruchamiany. Oznacza to trzy sprawy:

- Jeżeli dany proces ma część proceduralną, to interpreter rozpoczyna jej wykonywanie.
- Wszystkie podprocesy danego procesu zmieniają swój stan z „nieaktywny” na „oczekujący”. Może to być uwarunkowane od zakończenia części proceduralnej, ale niekoniecznie, można też traktować, że nadproces i jego podprocesy są wykonywane równolegle.
- Warunek uruchomionego procesu nie jest dalej badany. Proces można natomiast przerwać (poprzez odpowiednią autoryzowaną instrukcję zmieniającą jego stan) - w tym momencie przerwane są także wszystkie jego podprocesy. Przerwanie procesu nie oznacza jego usunięcia, usunięcie jest odrębną operacją regulowaną np. administracyjnie. Proces można go także zawiesić i odwiesić - w tym momencie są zawieszane/odwieszane wszystkie jego podprocesy, ale o ile były już kiedyś uruchomione, ich warunki początkowe nie są badane.

Idea jest taka, że jeżeli dany proces nie jest uruchomiony, to wszystkie jego podprocesy są w stanie „nieaktywny”, zaś ich warunki początkowe nie są badane. Dopiero wtedy, gdy dany proces będzie uruchomiony przez warunek początkowy, to wszystkie jego podprocesy znajdują się w stanie „oczekujący”, co oznacza, że ich warunki od tego momentu są badane przez interpreter i w zależności od wyniku badania podprocesy są uruchamiane lub nie. Dzieje się to oczywiście rekurencyjnie, w dół hierarchii procesów. Jeżeli dla kilku (pod-)procesów spełnione są ich warunki początkowe, to uruchamiane są one równolegle i niezależnie.

(8) Warunek końcowy procesu. Jest to element nieobowiązkowy. Dla procesów atomowych (nie posiadających podprocesów) nie musi wystąpić. W tym przypadku jego odpowiednikiem jest to, że sterowanie procesu opuściło kod, nie ma już nic do zrobienia. Podobnie, jeżeli dla procesów złożonych wszystkie podprocesy są zakończone i kod tego procesu nie jest już wykonywany, to ten proces uważamy za zakończony.

Analogicznie do warunku początkowego, warunek końcowy ustala, kiedy dany proces jest zakończony. Jak poprzednio, warunek jest badany cyklicznie, przy czym cykl badania jest ustalany przez projektanta procesu i zależy od skali czasowej danego biznesu. Jak poprzednio,

warunek końcowy jest dowolnym zapytaniem SBQL zwracającym *true/false*, odwołującym się do dowolnych globalnych i lokalnych informacji całości danego środowiska aplikacji workflow. Jeżeli warunek końcowy dla danego procesu jest *true*, to wszystkie jego uruchomione ale nie zakończone podprocesy są ustawiane w stan „przerwany” i nie kontynuowane, wszystkie nie uruchomione procesy są ustawiane w stan „nieaktywny”. Dany proces jest ustawiany w stan „zakończony” i nic się dalej z nim nie dzieje poza sprawami archiwalnymi (np. raportowaniem, data mining, etc.).

Warunek końcowy pełni jednocześnie rolę synchronizatora równoległych procesów. W warunku tym można np. zbadać, czy wszystkie składowe podprocesy ustawiły się w stan „zakończony”.

Zwróć uwagę na szczegół, który może prowadzić do wątpliwości. Chodzi o kod realizacji danego procesu, który ma podprocesy. Niekiedy ten kod rozpada się na dwie części: część realizowaną na wstępie, przed uruchomieniem podprocesów, oraz część realizowaną po zakończeniu, czyli po realizacji wszystkich podprocesów lub po ustawieniu warunku końcowego na *true*. Do wyboru są dwie strategie: albo rzeczywiście rozbić ten kod na dwie części: przed uruchomieniem podprocesów i po zakończeniu wszystkich podprocesów, albo ten kod zostawić w jednej części, ale wtedy potrzebna jest instrukcja *wait for <warunek>*, która zawiesza sterowanie do momentu, kiedy warunek jest *true*. Pierwsze rozwiązanie zwiększa nieco złożoność koncepcyjną całości. Drugie rozwiązanie może prowadzić do narzutów czasowych związanych z częstotliwością badania warunku.

(9) Powiązania tego procesu z innymi obiektami służą do połączenia procesu z informacjami zapisanymi w dowolnych innych obiektach. Przykłady takich powiązań są następujące:

- Powiązanie procesu z zasobem abstrakcyjnym (żądanym, zaplanowanym), np. rolą wykonawcy, abstrakcyjnym zasobem komputerowym (np. drukarką), itp.;
- Powiązanie procesu z zasobem konkretnym (obecnie używanym lub użytym), np. konkretnym wykonawcą aktualnie wykonującym przydzielone zadanie, konkretnym wykonawcą, który już wykonał zadanie, konkretną zasobem komputerowym (np. drukarką) przydzielonym do procesu lub użytym przez proces, itd.
- Powiązanie procesu z obiektem przechowującym jego definicję;
- Powiązanie procesu z jego typem (a la DTD lub XML Schema), który kontroluje jego strukturę;
- Powiązanie danego procesu ze słownikiem, który kontroluje wartość określonego atrybutu procesu;
- Powiązanie procesu z metadanymi (ontologią), które umożliwiają odszukanie procesu i dalsze operacje na podstawie pewnych cech jego „handlowych”;
- Powiązanie procesu z informacją ustalającą reguły bezpieczeństwa, np. reguły dostępu, prywatność, własność, itp.
- Powiązanie procesu z używanymi przez niego dokumentami, np. fakturą;
- Powiązanie procesu z dowolnymi danymi biznesowymi z otoczenia danej aplikacji workflow, np. stanem magazynu lub danymi marketingowymi;
- Powiązania procesu z używanymi przez niego aplikacjami;
- itd.

Generalnie, istnieją co najmniej 3 metody realizacji takich powiązań:

- Przez obiekt pointerowy, którego wartością jest referencja do innego obiektu;
- Przez pewną wartość (a la klucz obcy), która jest identyczna z wartością w innym obiekcie (a la klucz główny);
- Przez nazwę (+ścieżka) innego obiektu, np. nazwę i ścieżkę aplikacji zewnętrznej.

Powiązania mogą być oczywiście lokalne (prywatne dla danego procesu) i globalne (widoczne dla zapytań z zewnątrz). Powiązania mogą także wchodzić w skład atrybutów złożonych. Np. powiązanie do konkretnego wykonawcy może występować w obrębie atrybutu złożonego zawierającego również rolę wykonawcy, początek pracy, koniec pracy, ilość zużytego czasu, stawka godzinowa wynagrodzenia, itp.

(10) Powiązanie obiektu z klasami może być zaimplementowane w celu wykorzystania niezmienników obiektów znajdujących się w tych klasach (jak zwykle w obiektowości). Przyjmijmy, że klasa jest trwałym obiektem bazo-danowym znajdującym się po stronie serwera (jest trwała i jest pierwszej kategorii programistycznej). Podstawowymi niezmiennikami zapisanymi w klasie mogą być:

- Informacja typologiczna służąca do kontroli typologicznej obiektów oraz statycznej kontroli typologicznej zapytań/programów operujących na obiekcie;
- Kody metod zapisanych w ramach klasy (ich bindery są wrzucane na stos środowiskowy w momencie otwierania wnętrza obiektu);
- Wartości domyślne;
- Wspólne (statyczne) atrybuty;
- Kody warunków integralności, trygerów lub perspektyw przeciążających;
- Informacja dotycząca reguł bezpieczeństwa;
- Informacja dotycząca wizualizacji obiektu (np. grafika ikony wizualizacyjnej lub metoda wizualizacyjna);
- itd.

Klasy mogą być oczywiście powiązane w hierarchię dziedziczenia lub wielo-dziedziczenia, jak zwykle w obiektowości. Do rozważenia jest koncepcja dynamicznych ról, ale to może być również jakiś etap rozwoju.

(11) Proces może zawierać dowolną liczbę (pod-)procesów. Liczba poziomów hierarchii procesów nie jest ograniczona. W szczególności, mogą być procesy atomowe, nie zawierające pod-procesów. Każdy proces i podproces ma tę samą budowę i z punktu widzenia operacji generycznych systemu są traktowane tak samo (relatywizm procesów, analogiczny do relatywizmu obiektów w SBA). Nie wprowadzamy więc podziału na procesy, zadania i akcje (jak w innych propozycjach), chociaż taka terminologia może się niekiedy przydać, aby lepiej wyjaśnić sprawę dla osób z zewnątrz przyzwyczajonych do standardowej terminologii.

Proste i złożone procesy (zawierające pod-procesy) mogą powstawać w trojaki sposób:

- Przez instancjonowanie definicji procesu. W tym przypadku definicja procesu jest kopiowana, a następnie wszystkie miejsca określone przez parametry definicji są wypełniane właściwymi parametrami. Celem umożliwienia wprowadzenia procesu *ad hoc* powinna istnieć definicja pusta, która generuje pustą instancję

procesu. Tak pusta instancja może być zapełniona przez odpowiednie udogodnienie (mającego postać API).

- Poprzez „ręczne” wstawienie do środka danego procesu dowolnego pod-procesu. Może to być wstawienie instancji pustego podprocesu, a następnie zapełnienie go przy pomocy wspomnianego udogodnienia. Może to być także wstawienie zdefiniowanego procesu na podstawie skopiowania definicji oraz zapełnienie kopii właściwymi parametrami. Ręczna wstawienie pod-procesu może być związane z koniecznością modyfikacji warunku końcowego nad-procesu (inaczej ręcznie wstawiony proces może się nie wykonać lub może być w każdej chwili przerwany przez spełnienie warunku końcowego). Nie zawsze oczywiście, gdyż warunku końcowego może nie być (co oznacza konieczność zakończenia wszystkich podprocesów zanim zakończy się dany proces) lub warunek jest objęty kwantyfikatorem, funkcją agregowaną, itp. udogodnieniami języka zapytań.
- Poprzez refleksję, czyli wstawienie automatycznie wygenerowanych podprocesów do środka danego lub dowolnego innego identyfikowanego w systemie procesu. Sposób generacji tej definicji jest w zasadzie dowolny, pod-proces taki może nawet nie mieć swojej definicji. Jak poprzednio, automatyczna generacja podprocesu może oznaczać konieczność zmiany warunku końcowego danego procesu, nie zawsze oczywiście.

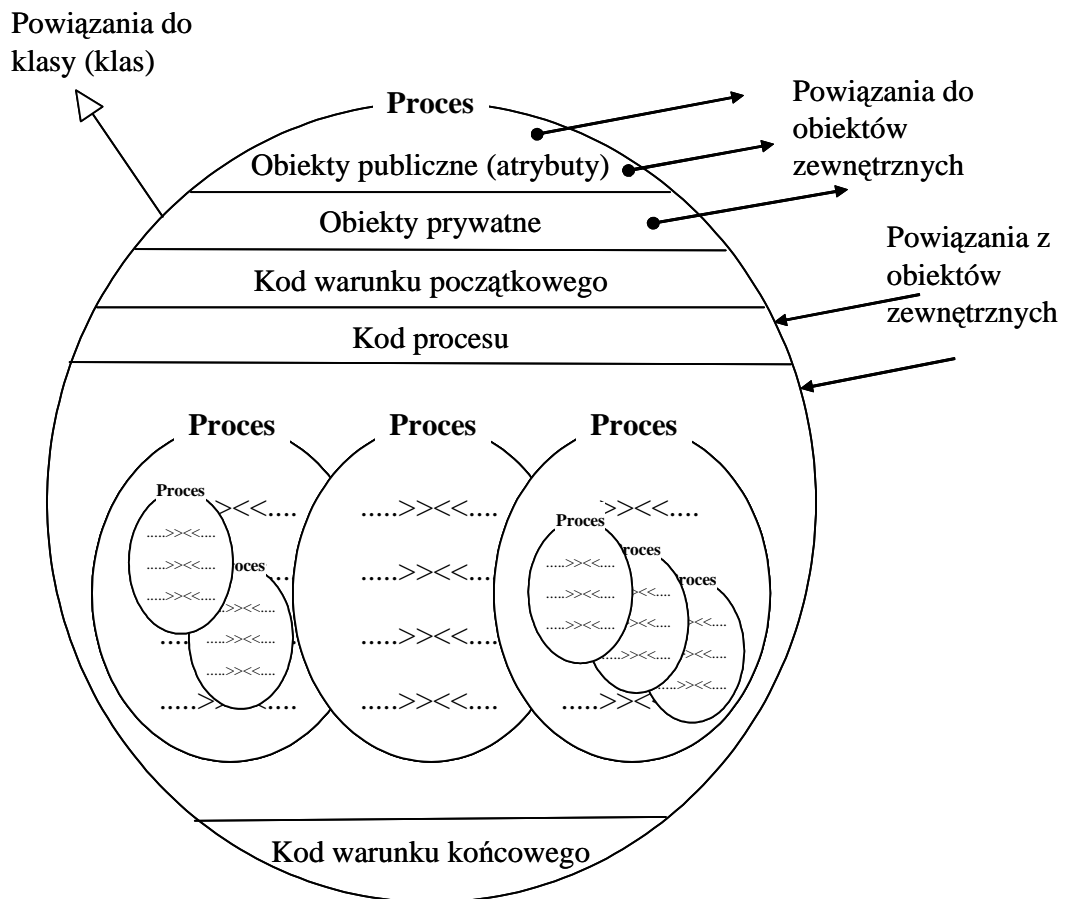
Analogicznie do wstawiania pod-procesów, może zachodzić ich usuwanie, również być może połączone ze zmianą warunku końcowego danego procesu.

„Ręczne” wstawianie lub refleksja mogą doprowadzić do niezgodności procesu z jego typem bądź definicją. Zakładamy więc, że dowolny proces może być jednego z czterech rodzajów:

- Proces nie związany z żadnym typem, który można dowolnie modyfikować poprzez wstawianie lub usuwanie podprocesów;
- Proces związany z typem, ale naruszenie typu jest dopuszczalne i ma konsekwencje tylko w postaci warknięcia (*warning*); taki proces można oczywiście dowolnie modyfikować;
- Proces związany z typem, ale naruszenie typu jest niedopuszczalne; taki proces można modyfikować, ale tylko w obrębie typu (np. jeżeli mamy kolekcję podprocesów tego samego typu, to można dorzucić jeszcze jeden pod-proces do tej kolekcji);
- Proces związany bądź nie związany z typem, którego jakiejkolwiek modyfikacje są niedopuszczalne. Tego rodzaju procesy będą wprowadzane tam, gdzie projektant nie dopuszcza modyfikacji ze względu na możliwość utraty spójności biznesowej procesu, doprowadzenie do zagrożenia bezpieczeństwa, złamania atomowości transakcji, itp.

Możliwe, że interesujące będą jeszcze inne przypadki, jako że pojęcie typu i kontroli typologicznej dla procesów jest cały czas zagadnieniem otwartym. Nawet nie jestem pewien, czy ktokolwiek podjął ten temat w literaturze.

Rys. 1 jest ilustracją podstawowych aspektów zagnieżdżonych procesów.



Rys.1 Poglądowy obraz zagnieżdżonego procesu

3. Dalsze tematy do opracowania w ramach tego projektu

Przewidujemy, że w toku prac opracowane będą dalsze tematy w ramach tej koncepcji:

- Definicja procesu i pod-procesów
- Zarządzanie czasem
- Wyjątki
- Zarządzanie zasobami
- Zarządzanie dokumentami
- Zarządzanie aplikacjami
- Metamodel
- Zmiany w instancjach procesów
- Refleksja
- Grupowanie i rozgrupowywanie instancji procesów i zadań
- Język zapytań/programowania, procedury, metody i perspektywy
- Współbieżność i transakcyjność
- Równoległe kontynuowanie procesów na innych serwerach lub na klientach
- Współpraca z wykonawcami zadań
- Kontrola typologiczna oraz integralnościowa procesów
- Grupowanie i rozgrupowywanie