Projekt Loglan - wczoraj, dziś, jutro

Andrzej Salwicki

salwicki@mimuw.edu.pl
Wydział Matematyczno-Przyrodniczy UKSW

16 października 2012
Wydział Informatyki

Politechniki Poznańskiej

Specyfikacja nowego języka - wybór pożądanych cech:

Specyfikacja nowego języka - wybór pożądanych cech:

Zadanie: zaprojektować i zrealizować język obiektowy o następujących cechach:

Specyfikacja nowego języka - wybór pożądanych cech:

Zadanie: zaprojektować i zrealizować język obiektowy o następujących cechach:

- efektywna i bezpieczna dealokacja niepotrzebnych obiektów,
- jednorodny model dla programowania współbieżnego i rozproszonego,
- w pełni obiektowy mechanizm komunikacji pomiędzy obiektami procesów,
- obiekty współprogramów kooperujące z procedurami rekurencyjnymi,

Specyfikacja nowego języka - wybór pożądanych cech:

Zadanie: zaprojektować i zrealizować język obiektowy o następujących cechach:

- efektywna i bezpieczna dealokacja niepotrzebnych obiektów,
- jednorodny model dla programowania współbieżnego i rozproszonego,
- w pełni obiektowy mechanizm komunikacji pomiędzy obiektami procesów,
- obiekty współprogramów kooperujące z procedurami rekurencyjnymi,

To kiepski żart. Taki język istnieje **od 30 lat**.

Nasz plan

1. Nieco historii

2. Ważniejsze wyniki
   - Kreczmara bezpieczny system zarządzania obiektami
   - Łączenie loglanowskich maszyn wirtualnych
   - Alien call
   - Porównanie języków programowania obiektowego

3. Problemy i zadania

Streszczenie

Tezy:

Streszczenie

Tezy:

1. Podczas prac nad Loglanem rozwiązano wiele istotnych problemów badawczych!

Streszczenie

Tezy:

1. Podczas prac nad Loglanem rozwiązano wiele istotnych problemów badawczych!
2. Żaden język programowania obiektowego nie osiągnął cech porównywalnych z Loglanem'82,

Streszczenie

Tezy:

1. Podczas prac nad Loglanem rozwiązano wiele istotnych problemów badawczych!
2. Żaden język programowania obiektowego nie osiągnął cech porównywalnych z Loglanem'82,
3. Praca nad nowym językiem LEM'12 może być kopalnią nowych problemów.

Wczoraj

Nieco historii?

Początki

W roku 1977, w Instytucie Maszyn Matematycznych MERA opracowaliśmy język Loglan'77.[SBHM77] Wynik był na tyle interesujący, że w r. 1978 podpisano umowę o dzieło: *Zaprojektowanie języka programowania Loglan*[1] *i realizację kompilatora tego języka na maszyny MERA 400*. Dzisiaj umowę taką nazwalibyśmy grantem. Zleceniodawcą było Zjednoczenie MERA - producent minikomputerów. Należy podkreślić zaangażowanie wicedyrektora naukowego zjednoczenia MERA, prof. Andrzeja Janickiego i jego wiarę w nasz sukces. Był on promotorem naszych poczynań.
Zleceniobiorcą został Instytut Informatyki UW, a konkretnie Zakład Teorii Obliczeń, którym wtedy kierowałem.

[1] dopiero parę lat później dowiedzieliśmy się, że dr C. Brown obmyślił i ogłosił własny esperanto-podobny język Loglan

Zespoły

Byliśmy trochę przerażeni wielkością zadania. Udało się namówić do pracy prawie wszystkich kolegów w Zakładzie. Naszym największym sukcesem było namówienie profesora Antoniego Kreczmara by pokierował pracami nad kompilatorem. W nieformalny sposób podzieliliśmy się na kilka zespołów:

- *Definicja języka Loglan* – {A. Salwicki, A. Kreczmar, T. Müldner, W.M. Bartol, H. Oktaba, A. Litwiniuk },
- *Kompilator* – {A. Kreczmar, D. Szczepańska, A. Litwiniuk, M. Lao, W. Nykowski},
- *Zespół rozbudowujący środowisko na Merze* – {P. Gburzyński, P. Findeisen},
- *Zespół wsparcia i programowania w Loglanie* – { T. Müldner, G. Mirkowska, L. Banachowski, A. Szałas, A. Salwicki, U. Petermann, ...}

Jak się rozliczyliśmy z umowy formalnej?

1. opublikowaliśmy Raport języka Loglan, PWN Warszawa, 1983
2. oddaliśmy kompilator wraz z pełną dokumentacją: Podręcznik uzytkownika, pliki źródłowe (1,5MB), etc.,
3. stworzyliśmy dla komputerów MERA 400: system plików, edytor, assembler, i innne narzędzia.
4. działaliśmy na rzecz wdrożenia jezyka:
   - 2 konferencje międzynarodowe (Zaborów[Zab1983], Radziejowice)
   - Jesienna Szkoła PTI Serock 1984[Ser1984].

   Wdrażanie w dydaktyce:
   - na politechnikach w Poznaniu i Białymstoku, Loglan był używany i wykładany do mniej więcej 2000 r.,
   - na UW trwało to krócej.

Współpraca międzynarodowa

- Institut für Informatik, Universität zu Kiel,
  Profesor Hans Langmaack
  - skorygował pewien błąd w naszej koncepcji statycznego wiązania identyfikatorów,
  - pomógł w przeniesieniu Loglanu na mainframe'y,
  - a ostatnio wziął udział w rozwiązywaniu problemu wyznaczania bezpośrednich superklas w Javie.
- IASI CNR Roma, dr Giana Cioni,
- Universita "*La Sapienza*", Rome, prof. Alfonso Miola
- i in. (Uniwersytety w Tübingen, Bordeaux, Caen, ...)

wkład doktorantów

1. O. Świda: VLP - wieloprocesorowy, rozproszony, wirtualny komputer loglanowski i środowisko,
2. A. Szałas: komunikacja procesów przez przerwania, ta koncepcja weszła do Loglanu'88 1983,
3. D. Szczepańska: system zgłaszania wyjątków i ich obsługi, 1982(wdrożenie), 1990 doktorat,
4. P. Gburzyński: System automatycznego dowodzenia twierdzeń zaprogramowany w Loglanie, realizacja koncepcji prof. M. Bibela, 1982 (Bibel miał swoją realizację 2 lata później)
5. H. Oktaba: formalizacja systemu zarządzania pamięcią 1982,
6. W.M. Bartol: opis systemu współprogramów, 1983
7. U. Petermann: komunikacja procesów przez przerwania
8. A. Litwiniuk - autor generatora kodu w kompilatorze,
   wraz z P. Gburzyńskim przenieśli Loglan na mainframe'y, później na maszyny VAX/VMS

Wkład studentów

- kompilator: parser – W. Nykowski 1981
- koncepcja i realizacja obcego wołania metod – B. Ciesielski 1988
- debugger – T. Przytycka 1984
- przeniesienie kompilatora
  - na IBM PC/AT – M. Benke i G. Grudziński 1985
  - do systemu Unix – P. Susicki 1989
  - na komputer Atari – Sebastien Bernard, Universite de Pau 1992
  - z 16 bitowego DOS na 32 system Windows95 – F. Pataud 1994
- środowisko Lotek – grupa studentów UW 1983
- klasy dla grafiki i obsługi myszki
  - XIIUWGRaph studenci z Universite de Pau, J. Larrieu, P. Becourt
  - IIUWGraph dla 32 bitowych PC – F. Pataud,
- wtyczka Loglanowska do Eclipse - A. Chwedoruk 2005
- kompilacja na Linuxa - A. Adamski, 2011
- wersja na platformę Windows: 7, XP - T.Reda 2011

Ta lista nie jest pełna.

razem

To co udało się osiągnąć, osiągnięto dzięki pracy zespołowej!

Dziś

Ważniejsze wyniki

Unikalne cechy języka Loglan'82

Unikalne cechy języka Loglan'82

- System bezpiecznego zarządzania pamięcią obiektów.

Unikalne cechy języka Loglan'82

- System bezpiecznego zarządzania pamięcią obiektów.
- Protokół *obcego wołania metod* (ang. *alien call*) w procesach.

Unikalne cechy języka Loglan'82

- System bezpiecznego zarządzania pamięcią obiektów.
- Protokół *obcego wołania metod* (ang. *alien call*) w procesach.
- Poprawny system współprogramów.

Unikalne cechy języka Loglan'82

- System bezpiecznego zarządzania pamięcią obiektów.
- Protokół *obcego wołania metod* (ang. *alien call*) w procesach.
- Poprawny system współprogramów.
- Klasy wewnętrzne i dziedziczenie ukośne.

Unikalne cechy języka Loglan'82

- System bezpiecznego zarządzania pamięcią obiektów.
- Protokół *obcego wołania metod* (ang. *alien call*) w procesach.
- Poprawny system współprogramów.
- Klasy wewnętrzne i dziedziczenie ukośne.
- Łączenie maszyn wirtualnych w wirtualny wieloprocesorowy komputer.

Unikalne cechy języka Loglan'82

- System bezpiecznego zarządzania pamięcią obiektów.
- Protokół *obcego wołania metod* (ang. *alien call*) w procesach.
- Poprawny system współprogramów.
- Klasy wewnętrzne i dziedziczenie ukośne.
- Łączenie maszyn wirtualnych w wirtualny wieloprocesorowy komputer.

powodują, że:

Unikalne cechy języka Loglan'82

- System bezpiecznego zarządzania pamięcią obiektów.
- Protokół *obcego wołania metod* (ang. *alien call*) w procesach.
- Poprawny system współprogramów.
- Klasy wewnętrzne i dziedziczenie ukośne.
- Łączenie maszyn wirtualnych w wirtualny wieloprocesorowy komputer.

powodują, że:

a) warto się zapoznać z Loglanem,

Unikalne cechy języka Loglan'82

- System bezpiecznego zarządzania pamięcią obiektów.
- Protokół *obcego wołania metod* (ang. *alien call*) w procesach.
- Poprawny system współprogramów.
- Klasy wewnętrzne i dziedziczenie ukośne.
- Łączenie maszyn wirtualnych w wirtualny wieloprocesorowy komputer.

powodują, że:

a) warto się zapoznać z Loglanem,

b) warto wykorzystać go, zwłaszcza w dydaktyce.

Unikalne cechy języka Loglan'82

- System bezpiecznego zarządzania pamięcią obiektów.
- Protokół *obcego wołania metod* (ang. *alien call*) w procesach.
- Poprawny system współprogramów.
- Klasy wewnętrzne i dziedziczenie ukośne.
- Łączenie maszyn wirtualnych w wirtualny wieloprocesorowy komputer.

powodują, że:

a) warto się zapoznać z Loglanem,

b) warto wykorzystać go, zwłaszcza w dydaktyce.

c) warto zapytać: *a gdzie są te cechy w moim ulubionym języku XYZ?*

Co oferujemy?

- kompilator i maszynę wirtualną na platformy Linux i Windows,
- podręczniki,
- wtyczkę do Eclipse,
- środowisko VLP z łatwą możliwością tworzenia wirtualnego, rozproszonego komputera loglanowskiego.

Niestety, nasz produkt nie jest opakowany komercyjnie.

Badania, które poprzedziły implementację Loglanu'82:

- Czy umożliwiając zagnieżdżanie klas i dziedziczenie można zachować mechanizm Display Vector znany z implementacji Algolu'60? [BKLH83],
- W jaki sposób wyznaczać bezpośrednią superklasę? [LSW08, LSW09], to nie jest banalne zadanie gdy zauważysz, że w jednym programie może być wiele klas o tej samej nazwie.
- Czy można stworzyć bezpieczny i efektywny system zarządzania pamięcią obiektów? Bez zjawiska wiszących referencji [MS87,].
- Czy można stworzyć system współprogramów (ang. coroutin) bardziej klarowny i wolny od sprzeczności systemu Simuli-67?
- Czy można stworzyć jedną platformę dla programowania współbieżnego i programowania rozproszonego?
- Czy istnieje obiektowy mechanizm komunikacji/synchronizacji dla programowania rozproszonego?
- itd.

Plan prezentacji

A. Kreczmara remedium na wiszące referencje

Aksjomat pamięci obiektowej

W każdym języku programowania obiektowego jego maszyna wirtualna (running system) musi zapewniać prawdziwość następującego twierdzenia:

Twierdzenie

Jeśli zmienna x jest zadeklarowana jako zmienna typu T to wartością tej zmiennej jest albo obiekt podklasy klasy T albo *none*.

$$type(x) = T \Rightarrow (x \texttt{ in } T \vee x = none)$$

Aksjomat pamięci obiektowej

W każdym języku programowania obiektowego jego maszyna wirtualna (running system) musi zapewniać prawdziwość następującego twierdzenia:

Twierdzenie

Jeśli zmienna x jest zadeklarowana jako zmienna typu T to wartością tej zmiennej jest albo obiekt podklasy klasy T albo *none*.

$$type(x) = T \Rightarrow (x \texttt{ in } T \vee x = none)$$

Aksjomat pamięci obiektowej

W każdym języku programowania obiektowego jego maszyna wirtualna (running system) musi zapewniać prawdziwość następującego twierdzenia:

Twierdzenie

Jeśli zmienna x jest zadeklarowana jako zmienna typu T to wartością tej zmiennej jest albo obiekt podklasy klasy T albo *none*.

$$type(x) = T \Rightarrow (x \texttt{ instanceof } T \vee x = null)$$

Aksjomat pamięci obiektowej

W każdym języku programowania obiektowego jego maszyna wirtualna (running system) musi zapewniać prawdziwość następującego twierdzenia:

Twierdzenie

Jeśli zmienna x jest zadeklarowana jako zmienna typu T to wartością tej zmiennej jest albo obiekt podklasy klasy T albo *none*.

$$type(x) = T \Rightarrow (x \texttt{ in } T \vee x = none)$$

Przypomnijmy

Definicja być podklasą

- Klasa T jest podklasą klasy T.
- Jeśli klasa Q rozszerza (`extends`) klasę R, i R jest podklasą klasy T, to Q jest podklasą klasy T.

Wyciek pamięci I

Z *wyciekiem pamięci* mamy do czynienia gdy podczas wykonywania programu rośnie liczba niepotrzebnych już obiektów.

Wyciek pamięci jest błędem, czasem jego konsekwencje są poważne. Co robić?

A) Jedne języki programowania zezwalają na usuwanie obiektu x (*free(x)* w Pascalu, *delete(x)* w C++, ...), jednak wiąże się z tym ryzyko pojawienia się wiszących referencji (*ang.* dangling references).

B) Inne języki (Java) nie dopuszczają takich instrukcji, tłumacząc, że odśmiecacz *gc()* (ang. garbage collector) usunie śmieci.

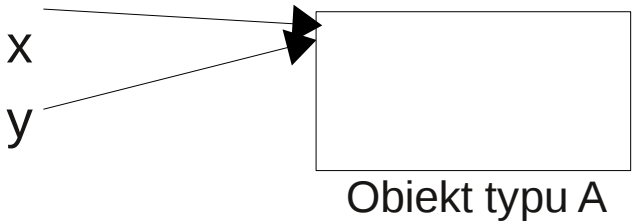
C) Loglan'82 ma
- odśmiecanie i
- usuwanie obiektów,
- a co najważniejsze, usuwanie obiektów jest bezpieczne.

Wyciek pamięci II

Błąd twórców Javy polega na tym, że zakładają, że obiekty niedostępne czyli *śmieci* i obiekty niepotrzebne to to samo, ale tak nie jest.
Czasami usuwanie obiektów niedostępnych nie zapobiega wyciekowi pamięci.
Natomiast programista może wiedzieć, o pewnym obiekcie, że choć jest dostępny (tj. nie jest śmieciem), to nie będzie używany w dalszym ciagu obliczeń. Warto by się go pozbyć.
W tej sytuacji – w Javie – programista **musi sam przekształcić niepotrzebny obiekt** *o* **w śmieć** usuwając wszystkie referencje do niego. A to nie jest ani łatwe, ani wolne od ryzyka.
Przy okazji: Odśmiecacz (ang. garbage collector) działa w czasie proporcjonalnym do rozmiaru pamięci. Ten rozmiar wzrósł ponad 1000 razy od czasu wprowadzenia Javy!
Może więc lepiej używać instrukcji kill(x)?

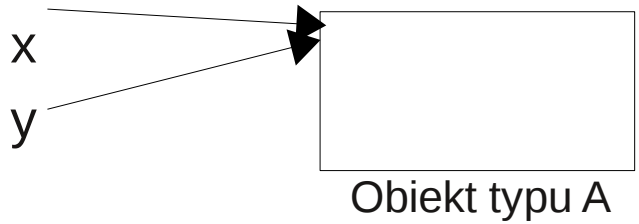
Wiszące referencje

Wiszące referencje



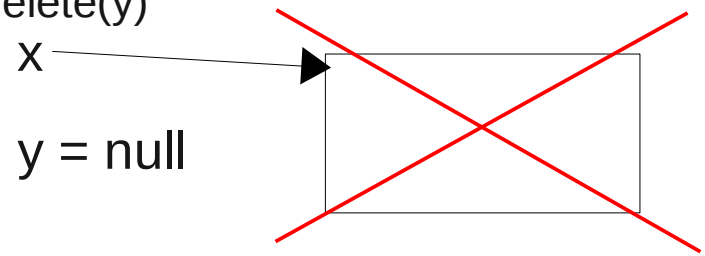
Częsta sytuacja

Wiszące referencje



Częsta sytuacja

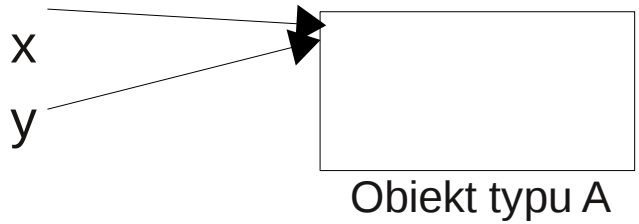
Po instrukcji delete(y)



Obiekt został zlikwidowany.
Wartość y=null jest w porządku, ale …
wartość x narusza aksjomat pamięci obiekto

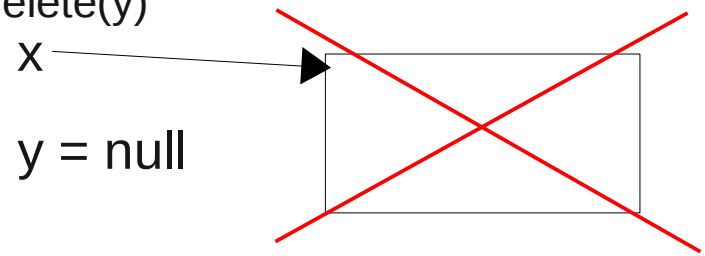
Pogwałcenie aksjomatu pamięci obiektowej!

Wiszące referencje



Częsta sytuacja

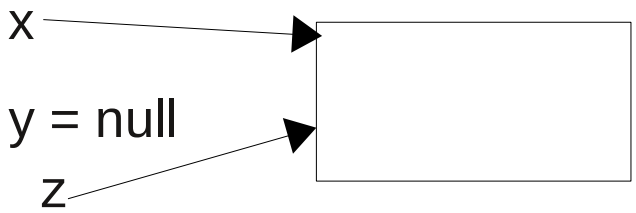
Po instrukcji delete(y)



Obiekt został zlikwidowany.
Wartość y=null jest w porządku, ale …
wartość x narusza aksjomat pamięci obiekto

Pogwałcenie aksjomatu pamięci obiektowej!

z := new B(...)

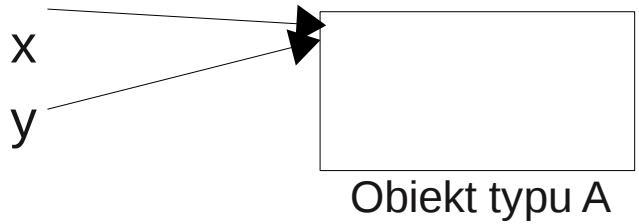


Tak też może się zdarzyć.

Zmienna x „mówi” tu jest obiekt typu A,
a zmienna z: tu jest obiekt typu B

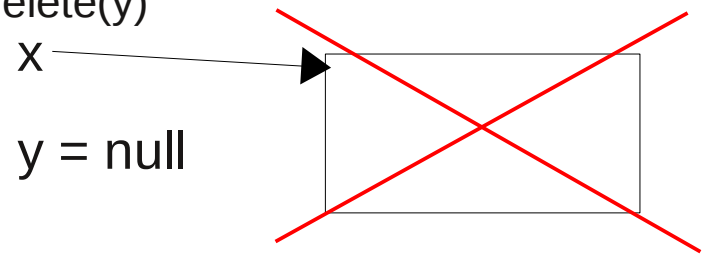
Sprzeczność!

Wiszące referencje



Częsta sytuacja

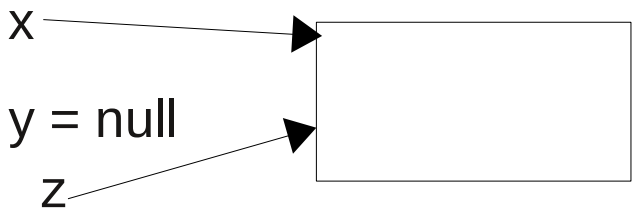
Po instrukcji delete(y)



Obiekt został zlikwidowany.
Wartość y=null jest w porządku, ale …
wartość x narusza aksjomat pamięci obiekto

Pogwałcenie aksjomatu pamięci obiektowej!

z := new B(...)



Tak też może się zdarzyć.

Zmienna x „mówi” tu jest obiekt typu A,
a zmienna z: tu jest obiekt typu B

Sprzeczność!

Tw. Nie istnieje algorytm wykrywania wiszących referencji!

Bezpieczne usuwanie obiektów

[2]zob. Algorithmic Logic pp. 328-341

[3]Warto pamiętać, że w takiej sytuacji próba wykorzystania metody m tego obiektu (np call x_2.m) lub bezpośredniego dostępu do pewnego atrybutu tego obiektu (np. x_3.z) zostanie wykryta, tj. zakończy się podniesieniem sygnału błędu.

Bezpieczne usuwanie obiektów

Prof. Antoni Kreczmar (∗1945 – †1996) zaprojektował i zrealizował kompletny i bezpieczny system zarządzania pamięcia obiektów (*ang.* heap).

[2]zob. Algorithmic Logic pp. 328-341

[3]Warto pamiętać, że w takiej sytuacji próba wykorzystania metody m tego obiektu (np call x_2.m) lub bezpośredniego dostępu do pewnego atrybutu tego obiektu (np. x_3.z) zostanie wykryta, tj. zakończy się podniesieniem sygnału błędu.

Bezpieczne usuwanie obiektów

Prof. Antoni Kreczmar (∗1945 – †1996) zaprojektował i zrealizował kompletny i bezpieczny system zarządzania pamięcia obiektów (*ang.* heap).
W systemie Kreczmara [2] zapewniono prawdziwość następującej formuły {schematu formuł}:

Niezmiennik systemu Loglan

$$(x_1 = \cdots = x_n \neq none) \Rightarrow [\text{kill}(x_i)](x_1 = \cdots = x_n = none)$$

Formuła ta tłumaczy się w ten sposób: jeśli jakiś obiekt jest wartością n zmiennych $x_1, x_2, \ldots, x_n$ to po wykonaniu instrukcji $\text{kill}(x_i)$ wszystkie te zmienne przyjmują wartość none. [3]

[2] zob. Algorithmic Logic pp. 328-341

[3] Warto pamiętać, że w takiej sytuacji próba wykorzystania metody m tego obiektu (np call x_2.m) lub bezpośredniego dostępu do pewnego atrybutu tego obiektu (np. x_3.z) zostanie wykryta, tj. zakończy się podniesieniem sygnału błędu.

Porównanie

Założenie: pewien obiekt o jest wartością zmiennych $x_1, x_2, \ldots, x_n$.
Zadanie polega na usunięciu obiektu o.

w Loglanie	**w Javie**
$kill(x_i)$	$x_1 = null;$ $x_2 = null;$ $\vdots$ $x_n = null;$ $gc();$
KOSZT:	
ok. 80 cykli	ok. miliona cykli

Ale nie w koszcie jest problem! Problem polega na tym by programujący w Javie **nie zapomniał o żadnej**(!) zmiennej x_i, która wskazuje na niepotrzebny już obiekt.

Klasy wewnętrzne i dziedziczenie ukośne

W języku SIMULA67 – matce wszystkich języków obiektowych – klasy mogą być zagnieżdżane. Ale dziedziczyć możemy tylko z klasy, która jest bratem danej klasy. Dziedziczenie w SIMULI67 jest *poziome*. W konsekwencji, język SIMULA nie dopuszcza do stworzenia biblioteki klas.
W Loglanie'82 zgodziliśmy się na dziedziczenie z klasy, która jest widoczna, np. jest zadeklarowana na wyższym poziomie zagnieżdżania klas. W związku z tym trzeba było rozwiązać kilka problemów:

- czy można zachować mechanizm dostępu do wielkości nielokalnych, znany jako Display Vector?
- w jaki sposób wyznaczać klasę z której dana klasa ma dziedziczyć? Zauważ, że teraz w programie może wystąpić wiele klas o takiej samej nazwie.

Problemy te udało się rozwiązać.
W Javie drugi z tych problemów jest znacznie trudniejszy – opublikowaliśmy poprawny i kompletny algorytm rozwiązujący ten problem(dla Javy).(LSW2009, LSW2008)

Współprogramy, ang coroutines I

Ten mechanizm jest znany od prawie 50 lat. Był zrealizowany w SIMULI67 i w Loglanie'82. Został nieco zapomniany i dopiero od niedawna zyskuje na popularności. Obliczenia z współprogramami możemy określić jako obliczenia quasi-współbieżne.
Współprogramy w SIMULI oceniono bardzo wysoko, m. in. D. Knuth. Ale ich opis nie był całkiem klarowny, a A. Wang wykazał, że zawiera on sprzeczność. System współprogramów został powiązany z blokiem prefiksowanym w niezbyt precyzyjny sposób.
W Loglanie uproszczono to co niezbyt jasno opisano w SIMULI. Obiekt współprogramu zawiera quasi-wątek, lub włókno (ang. fiber). W systemie quasi-współbieżnym conajwyżej jeden quasi-wątek jest aktywny. Instrukcje $attach(x)$ przenoszą sterowanie do łańcucha dynamicznego metod zaczynającego się w obiekcie współprogramu x.

Współprogramy, ang coroutines II

W wielu językach programowania stosowana jest bezparametrowa instrukcja yield(), w tych językach nie wiadomo, który współprogram zostanie wznowiony, decyzja należy do systemu, nie do programisty.

Łączenie drzew binarnych poszukiwań I

Zadanie polega na połączeniu zawartości pewnych drzew binarnych poszukiwań w ciag niemalejący.
Pierwszy pomysł jaki się nasuwa to powtarzanie
dopóki choć jedno drzewo jest niepuste
powtarzaj
  dla każdego drzewa
    znajdź element najmniejszy
    wydrukuj najmniejszy z nich
    usuń ten element z jego drzewa
  koniec dla
zakończ powtarzanie
Koszt takiego algorytmu jest raczej za duży. Każdy element tych drzew będzie oglądany wielokrotnie.

Łączenie drzew binarnych poszukiwań II

Węzły drzewa BST są obiektami klasy node:

```
unit node: class(value: integer);
   var lewy, prawy: node;
   unit insert: procedure(val: integer) ... end insert;
   unit ismember: function(val: integer): Boolean; ... end ismember;
   unit delete: procedure(val:integer); ... end delete;
end node;
```

Łączenie drzew binarnych poszukiwań III

```
unit traverse: procedure(n: node);
   begin
   if n =/= none then
   call traverse(n.lewy); drukuj(n.value); call traverse(n.prawy)
   endif
end traverse
```

Łączenie drzew binarnych poszukiwań IV

Ole-Johan Dahl i Arne Wang zaproponowali użycie współprogramów. Dla każdego drzewa d tworzymy współprogram W_d wyposażony w metodę *traverse*, której zadaniem jest odwiedzenie drzewa w porządku inorder.

```
unit W : coroutine(root: node);
   var val: integer;
   unit T: procedure(y : node);
   begin
      if y =/= none then
         call T(y.left); val := y.val; detach; call T(y.right);
      fi
   end T;
begin (* konstruktor dla W jest pusty *)
   return; (* tu rozpoczynamy po uaktywnieniu przez attach() *)
   call T(root); (* a gdyby tu wywołano traverse ...?*)
   val := Maximal;
end W;
```

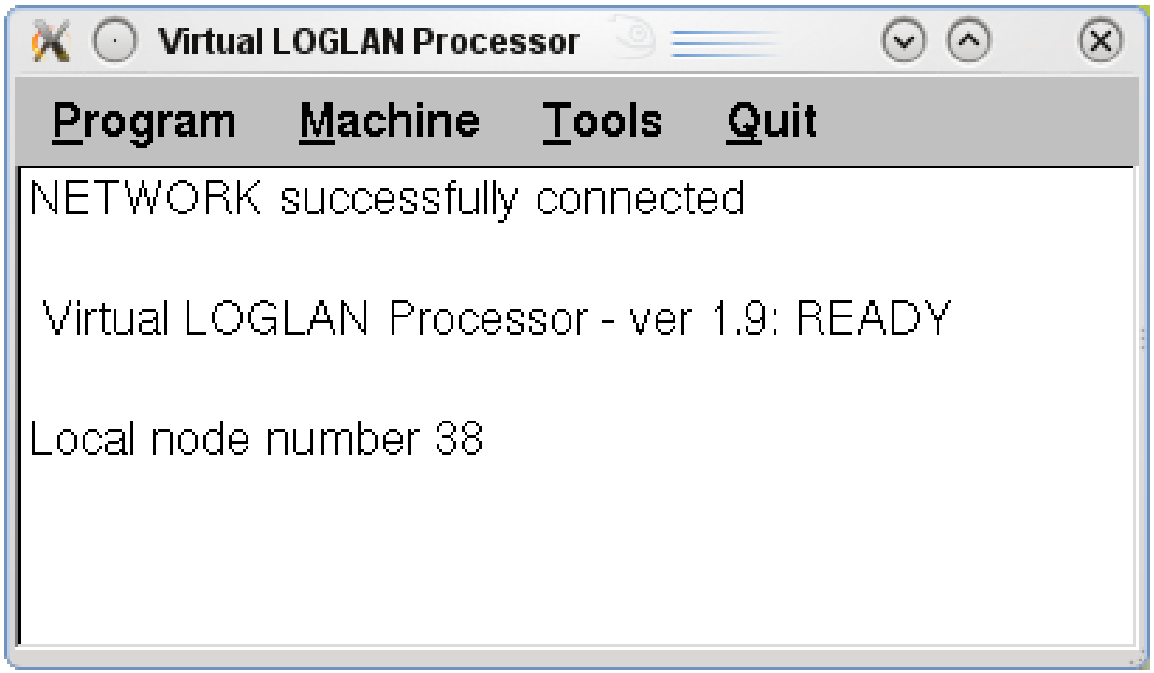
Łączenie drzew binarnych poszukiwań V

```
program test;
  var A: CA, B: CB, kolejny, n: integer, rt: node;
  unit CA: coroutine(n: node);
    unit T: procedure(y: node);
    begin
      if y<>none  then
          call T(y.left);
          kolejny := y.val; detach;
         call T(y.right)
      fi
    end T;
  begin return;call T(n);
  end CA;

  unit CB: coroutine;
  begin
      return;
     do
        attach(A);
        write(kolejny);
      od
  end CB

begin
    rt := zbudujBST;  A:= new CA(rt); B :=new CB; attach(B);
end
```

Plan prezentacji

Łączenie loglanowskich maszyn wirtualnych

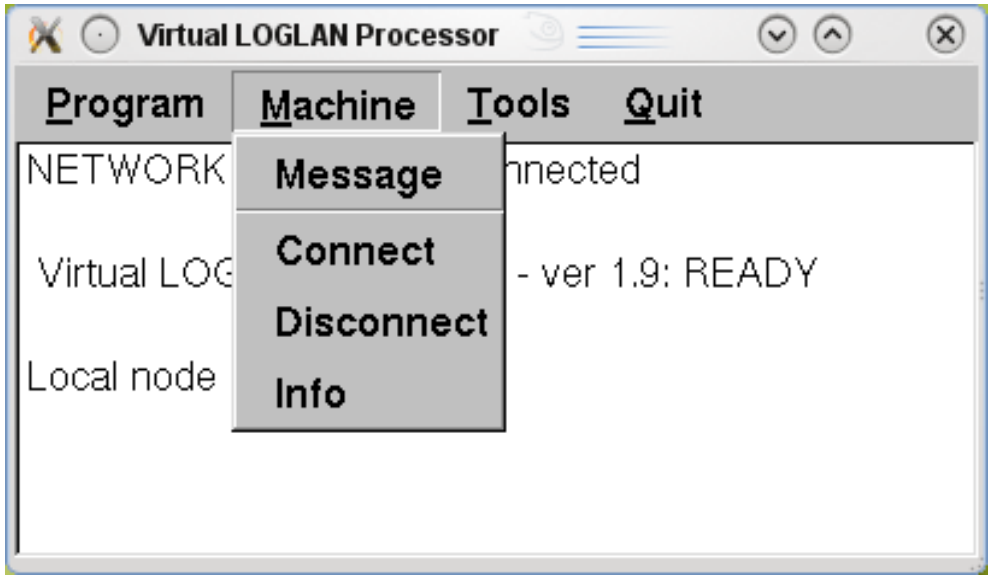
Oskar Świda stworzył środowisko do pracy z programami w Loglanie VLP – wirtualny procesor loglanowski.



Kilka slajdów ilustrujących działanie VLP

Łączenie loglanowskich maszyn wirtualnych

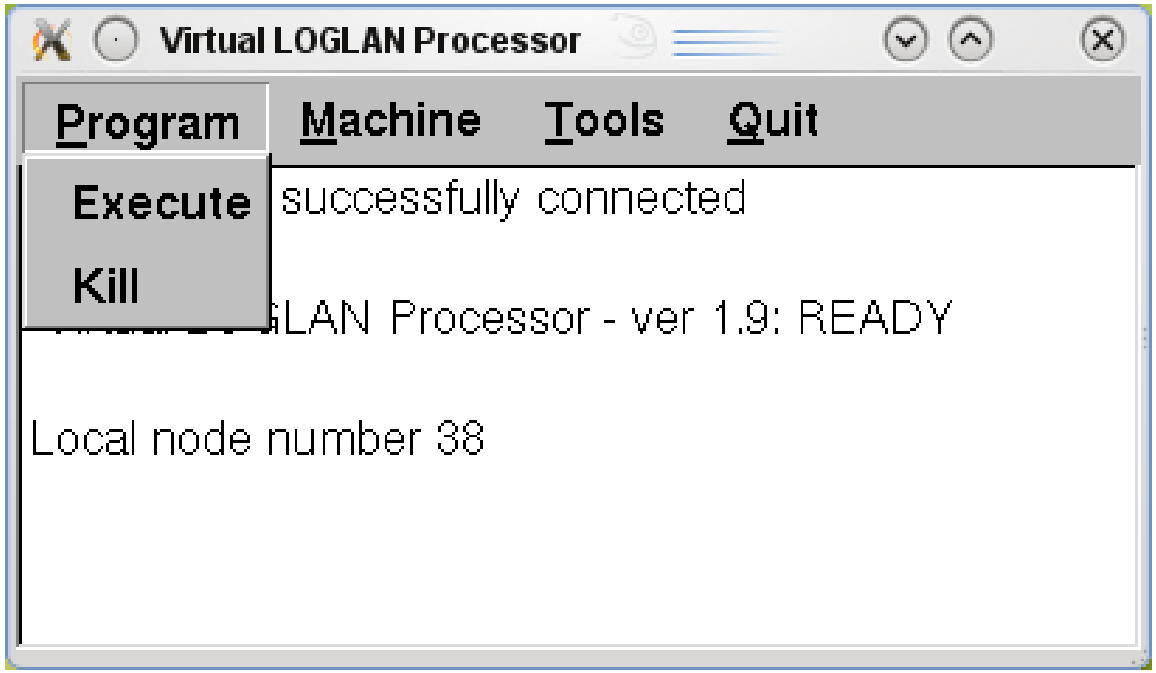
Jeżeli środowisko VLP jest otwarte na kilku lub więcej komputerach to mozemy je połączyć i uzyskać rozproszony loglanowski komputer wirtualny



Wybieramy polecenie connect i podajemy adres IP komputera z którym chcemy się połączyć.
Możesz także wpisać adresy wielu maszyn do pliku konfiguracyjnego 'vlp.cfg'. Wtedy uruchomienie VLP na Twoim komputerze zapewni

Łączenie loglanowskich maszyn wirtualnych

Okienko VLP jest konsolą, z której można uruchamiać programy (i je zabijać, gdy wymkną Ci się spod kontroli)



Powtarzam, to nie jest produkt komercyjny, to jest wynik badań dotyczących pytania: *jak rozdzielać obliczenia obiektowe w sieci?*
Ale można go rozwinąć i ulepszyć.
Pomożecie?

Plan prezentacji

Alien call

Klasy, współprogramy, procesy

W Loglanie mamy trzy rodzaje modułów (ang. unit):

Klasy, współprogramy, procesy

W Loglanie mamy trzy rodzaje modułów (ang. unit):

- klasy – **unit** C: **class** ... **end** C;

Klasy, współprogramy, procesy

W Loglanie mamy trzy rodzaje modułów (ang. unit):

- klasy – **unit** C: **class** ... **end** C;
- współprogramy – **unit** W: **coroutine** ... **end** W;

Klasy, współprogramy, procesy

W Loglanie mamy trzy rodzaje modułów (ang. unit):

- klasy – **unit** C: **class** ... **end** C;
- współprogramy – **unit** W: **coroutine** ... **end** W;
- procesy – **unit** P: **process** ... **end** P;

Klasy, współprogramy, procesy

W Loglanie mamy trzy rodzaje modułów (ang. unit):

- klasy – **unit** C: **class** ... **end** C;
- wspólprogramy – **unit** W: **coroutine** ... **end** W;
- procesy – **unit** P: **process** ... **end** P;

Operacja **new** powołuje do życia odpowiednio:obiekty klas, obiekty współprogramów i obiekty procesów (inaczej obiekty aktywne). Scenariusze tych obiektów są istotnie różne:

Klasy, współprogramy, procesy

W Loglanie mamy trzy rodzaje modułów (ang. unit):

- klasy – **unit** C: **class** ... **end** C;
- wspólprogramy – **unit** W: **coroutine** ... **end** W;
- procesy – **unit** P: **process** ... **end** P;

Operacja **new** powołuje do życia odpowiednio:obiekty klas, obiekty współprogramów i obiekty procesów (inaczej obiekty aktywne).
Scenariusze tych obiektów są istotnie różne:

- Obiekt klasy po utworzeniu pozostaje pasywny.

Klasy, współprogramy, procesy

W Loglanie mamy trzy rodzaje modułów (ang. unit):

- klasy – **unit** C: **class** ... **end** C;
- wspólprogramy – **unit** W: **coroutine** ... **end** W;
- procesy – **unit** P: **process** ... **end** P;

Operacja **new** powołuje do życia odpowiednio:obiekty klas, obiekty współprogramów i obiekty procesów (inaczej obiekty aktywne).
Scenariusze tych obiektów są istotnie różne:

- Obiekt klasy po utworzeniu pozostaje pasywny.
- Obiekt procesu po utworzeniu jest w stanie *pasywny*, lecz można go uaktywnić wykonując polecenie $resume(x)$.

$$(Active = S \& Passive = Q) \Rightarrow [resume(x)](Active = S \cup \{x\} \& Passive = Q \setminus \{x\})$$

Klasy, współprogramy, procesy

W Loglanie mamy trzy rodzaje modułów (ang. unit):

- klasy – **unit** C: **class** ... **end** C;
- wspólprogramy – **unit** W: **coroutine** ... **end** W;
- procesy – **unit** P: **process** ... **end** P;

Operacja **new** powołuje do życia odpowiednio:obiekty klas, obiekty współprogramów i obiekty procesów (inaczej obiekty aktywne).
Scenariusze tych obiektów są istotnie różne:

- Obiekt klasy po utworzeniu pozostaje pasywny.
- Obiekt procesu po utworzeniu jest w stanie *pasywny*, lecz można go uaktywnić wykonując polecenie $resume(x)$.

$$(Active = S \& Passive = Q) \Rightarrow [resume(x)](Active = S \cup \{x\} \& Passive = Q \setminus \{x\})$$

- Obiekty współprogramów mogą sobie przekazywać aktywność (tj. procesor) wykonując instrukcję $attach(x)$.

Niezmiennie $card(ActiveCoroutines) = 1$

Warstwy zarządzania obiektami

W trakcie obliczeń programu loglanowskiego może powstać wiele kooperujących obiektów procesów. Obiekty te mogą być alokowane i wykonywane na jednym procesorze loglanowskim (daje to obliczenia współbieżne), bądź na wielu procesorach połączonych siecią (obliczenia rozproszone), bądź też inną obmyśloną miksturę tych dwu klas obliczeń.

Każdy obiekt procesu może rządzić wieloma obiektami współprogramów – a więc może zarządzać obliczeniami quasi-współbieżnymi.

Każdy taki obiekt może tworzyć i zarządzać obiektami klas.

Wywołanie metody w obiekcie vs. obce wywolanie metody

Niech *o* będzie obiektem jakiejś klasy. Instrukcja

call *o.meth(params)*

powoduje, że program (proces) zaczyna wykonywać metodę *meth* z obiektu *o*. To jest wywołanie metody *meth* w obiekcie *o*.

Wywołanie metody w obiekcie vs. obce wywolanie metody

Niech o będzie obiektem jakiejś klasy. Instrukcja

call $o.meth(params)$

powoduje, że program (proces) zaczyna wykonywać metodę $meth$ z obiektu o. To jest wywołanie metody $meth$ w obiekcie o.

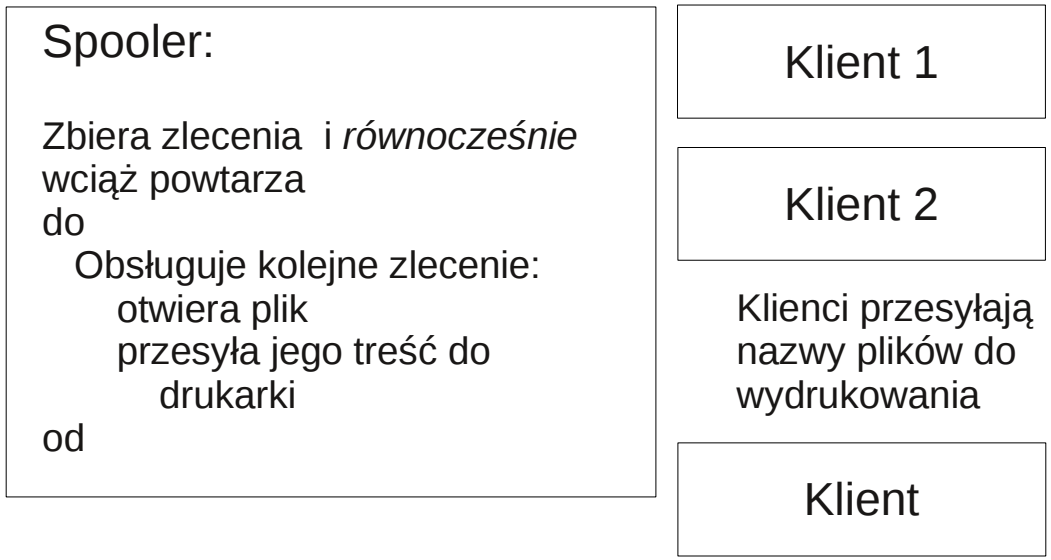
Jeśli obiekt p został utworzony na podstawie pewnego procesu, tj. gdy p jest obiektem aktywnym, to tak samo wyglądająca instrukcja wykonywana w aktywnym obiekcie q

call $p.mtd(parametry)$

ma zupełnie inne znaczenie – nieformalnie możemy ją opisać tak: obiekt aktywny q zwraca się do obiektu aktywnego p z prośbą o wykonanie metody mtd z dostarczonymi przez obiekt q parametrami $parametry$. To jest *obce wywołanie* metody mtd. Obiekt wzywany q może się zgodzić na przyjęcie takiej prośby wykonując instrukcję accept mtd bądź enable mtd lub ją zablokować wykonując instrukcję disable mtd.

Spooler

Pewna liczba klientów wysyła od czasu do czasu zlecenia wydrukowania pliku do serwera.



Serwer o nazwie spooler ma dwa zadania:
a) przyjmować zlecenia,
b) obsługiwać je w kolejności zgłoszeń.

Spooler 2

Spooler ma kolejkę Q i metodę *rejestruj*. Klienci rejestrują swoje zgłoszenia.
Wątek spoolera nie zajmuje się rejestrowaniem zadań.

```
unit spooler: process …
  var Q: kolejka;
  rejestruj: procedure(f: string);
     Wstaw f  do kolejki Q
  end rejestruj
begin
    Weź pierwszego z kolejki Q;
    Obsłuż
    ...
    ...
    i na drukarkę
end Spooler;
```

Kolejka: class;

Spooler 3

```
unit spooler: process ...
  var Q: kolejka;
   rejestruj: procedure(f: string);
      Wstaw f  do kolejki Q
   end rejestruj
begin
    Weź pierwszego z kolejki Q;
    Obsłuż
    ...
    ...
    i na drukarkę
end Spooler;
```

Kolejka: class;

Operacje *wstaw* i *weź* grożą konfliktem.

Zlecenia Q.*wstaw(...)* mogą nadchodzić (od klientów) przez cały czas.

Spooler 4

```
Spooler: proces(node: integer);
   var Q: kolejka;
   rejestruj: procedure(f: string);
begin
  do
      disable rejestruj;
        if Q.empty() then
           accept rejestruj   fi;
        plk := Q.first();
      enable rejestruj;
     (* wyślij plk na drukarkę *)
  od
end Spooler;
```

```
rejestruj:procedure(f:string);
begin
    call Q.insert(f);
    if Q.full() then
       return disable rejestruj
    fi;
end rejestruj;
```

Spooler 4

```
Spooler: proces(node: integer);
   var Q: kolejka;
   rejestruj: procedure   ...
begin
  do
      disable rejestruj;
        if Q.empty() then
           accept rejestruj   fi;
        plk := Q.first();
      enable rejestruj;
     (* wyślij plk na drukarkę *)
  od
end Spooler;
```

```
rejestruj:procedure(f:string);
begin
    call Q.insert(f);
    if Q.full() then
       return disable rejestruj
    fi;
end rejestruj;
```

Zapobiegamy konfliktom: klient nie może rejestrować podczas wybierania pliku z kolejki.

Tworzymy system klientów ze spoolerem i puszczamy go w ruch

```
program zlecenia_do_spoolera;
   spooler: process(node: integer); … end spooler;
   klient: process(node: integer, s:spooler); … end klient;
   var sp: spooler, k1, k2, k3, k4: klient
begin
    sp := new spooler(0);
    k1 := new klient(11, sp);
    k2 := new klient(25, sp);
    k3 := new klient(0, sp);
    k4 := new klient(11, sp);
    resume(sp); resume(k1);  …  resume(k4);
end
```

Pięć obiektów aktywnych umieścimy na trzech wirtualnych procesorach loglanowskich i następnie puścimy je w ruch. Mamy tu i rozpraszanie obliczeń i obliczenia współbieżne.

Własności spoolera

Zakładamy, że instrukcje drukowania wybranego pliku kończą się (bez zawieszenia i bez zapętlenia).

Twierdzenie

(i) Żadne zlecenie nie zostanie zgubione.
(ii) Obsługa zleceń nie dozna uszczerbku z powodu konfliktu zleceń bądź konfliktu zlecenie/obsługa zlecenia.

Nie może dojść do zapisania dwu plików w to samo miejsce w kolejce Q.
Nie może dojść do zgubienia pliku z powodu przepełnienia kolejki.

Dowód

Dowód

- Podczas wybierania pliku z kolejki nie dochodzi do nowej rejestracji.
  Wykonano `disable rejestruj`.

Dowód

- Podczas wybierania pliku z kolejki nie dochodzi do nowej rejestracji.
  Wykonano `disable rejestruj`.
- Podczas rejestracji wątek spoolera jest zawieszony (nastąpiło przerwanie).

Dowód

- Podczas wybierania pliku z kolejki nie dochodzi do nowej rejestracji.
  Wykonano `disable rejestruj`.
- Podczas rejestracji wątek spoolera jest zawieszony (nastąpiło przerwanie).
- Podczas rejestracji inny klient musi oczekiwać na jej zakończenie.
  Zbiór ENABLED *jest pusty*.

Dowód

- Podczas wybierania pliku z kolejki nie dochodzi do nowej rejestracji.
  Wykonano `disable rejestruj`.
- Podczas rejestracji wątek spoolera jest zawieszony (nastąpiło przerwanie).
- Podczas rejestracji inny klient musi oczekiwać na jej zakończenie.
  Zbiór ENABLED *jest pusty*.
- Po zapełnieniu kolejki kolejni klienci oczekują na miejsce w kolejce.
  Gdy po wpisaniu nazwy pliku do kolejki jest ona pełna, dokonuje się powrotu, który *nie odtwarza zbioru* ENABLED.

Dowód

- Podczas wybierania pliku z kolejki nie dochodzi do nowej rejestracji.
  Wykonano `disable rejestruj`.
- Podczas rejestracji wątek spoolera jest zawieszony (nastąpiło przerwanie).
- Podczas rejestracji inny klient musi oczekiwać na jej zakończenie.
  Zbiór ENABLED *jest pusty*.
- Po zapełnieniu kolejki kolejni klienci oczekują na miejsce w kolejce.
  Gdy po wpisaniu nazwy pliku do kolejki jest ona pełna, dokonuje się powrotu, który *nie odtwarza zbioru* ENABLED.
- Gdy kolejka jest pusta wątek oczekuje na zarejestrowanie jakiegoś pliku.
  Wykonanie `accept` powoduje oczekiwanie na rejestrację.

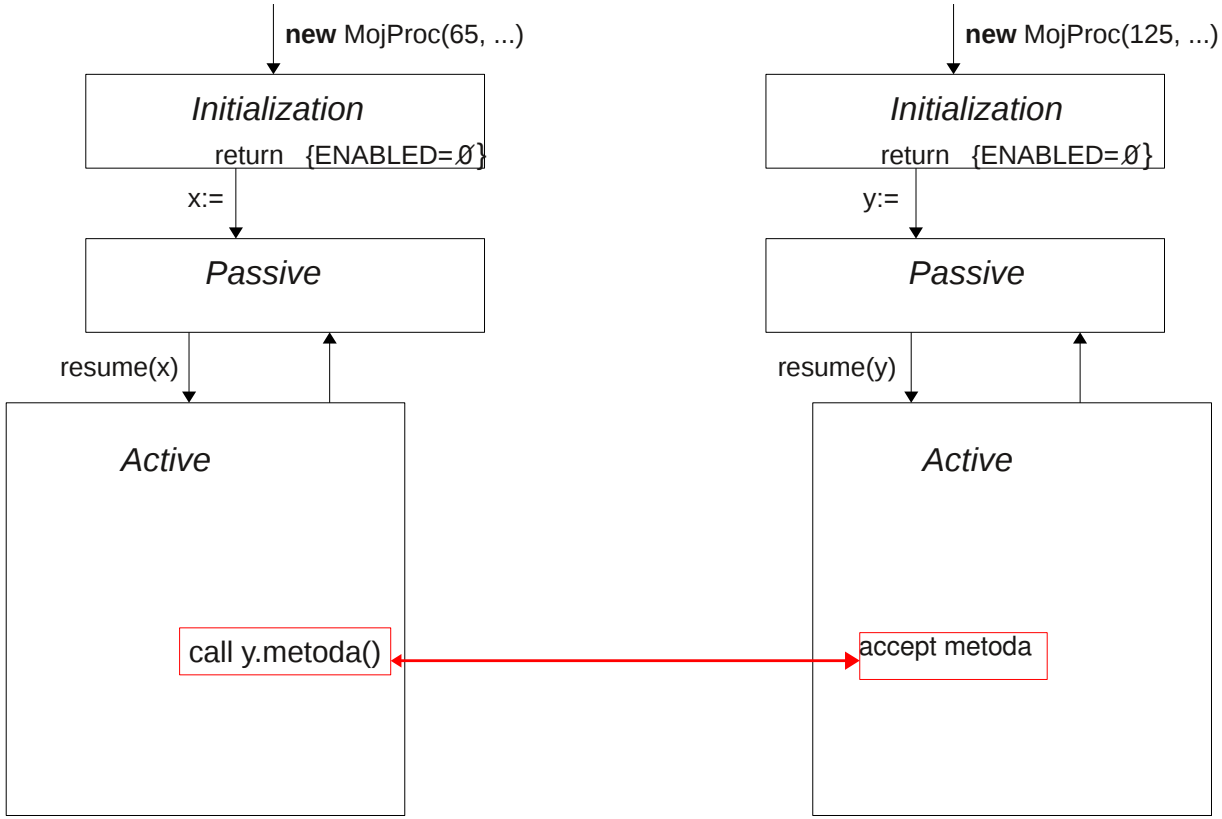
Dowód

- Podczas wybierania pliku z kolejki nie dochodzi do nowej rejestracji.
  Wykonano `disable rejestruj`.
- Podczas rejestracji wątek spoolera jest zawieszony (nastąpiło przerwanie).
- Podczas rejestracji inny klient musi oczekiwać na jej zakończenie.
  Zbiór ENABLED *jest pusty*.
- Po zapełnieniu kolejki kolejni klienci oczekują na miejsce w kolejce.
  Gdy po wpisaniu nazwy pliku do kolejki jest ona pełna, dokonuje się powrotu, który *nie odtwarza zbioru* ENABLED.
- Gdy kolejka jest pusta wątek oczekuje na zarejestrowanie jakiegoś pliku.
  Wykonanie `accept` powoduje oczekiwanie na rejestrację.
- Podczas opracowywania pliku i przesyłania go do drukarki może dokonywać się bezkonfliktowa rejestracja zgłoszeń. Operacja rejestruj *przerywa na chwilę* wątek spoolera.

Alien call I

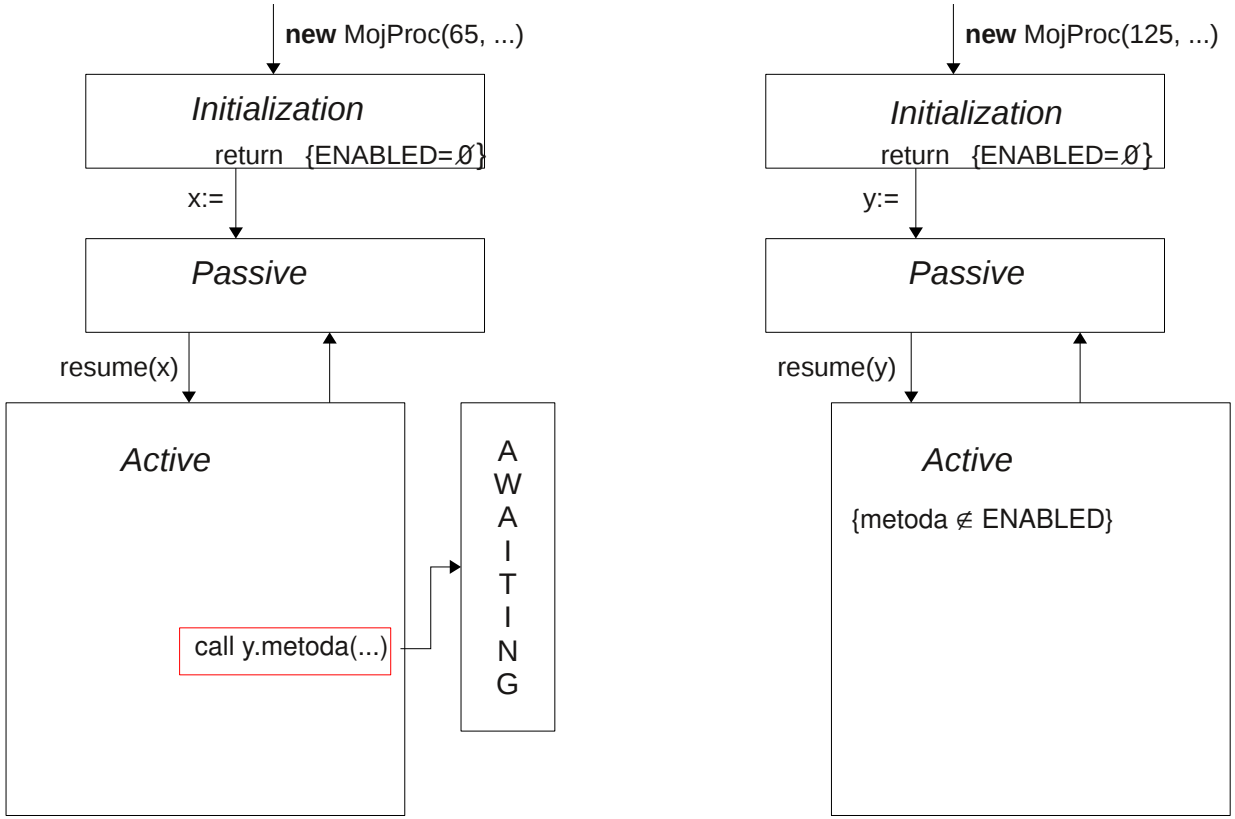
Objaśnimy protokół *alien call* na obrazkach.

Alien call II



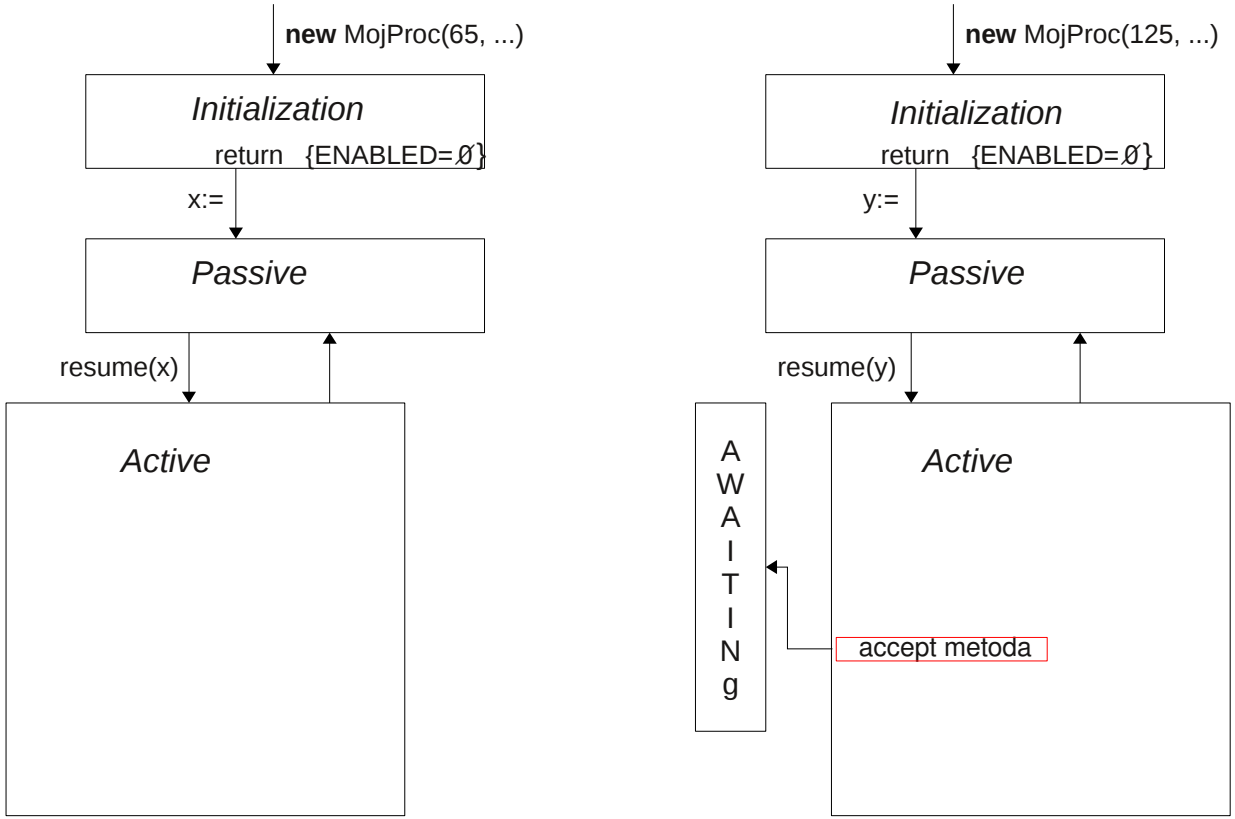
Rysunek: Protokół alien call przypadek 1

Alien call III



Rysunek: Protokół alien call przypadek 2

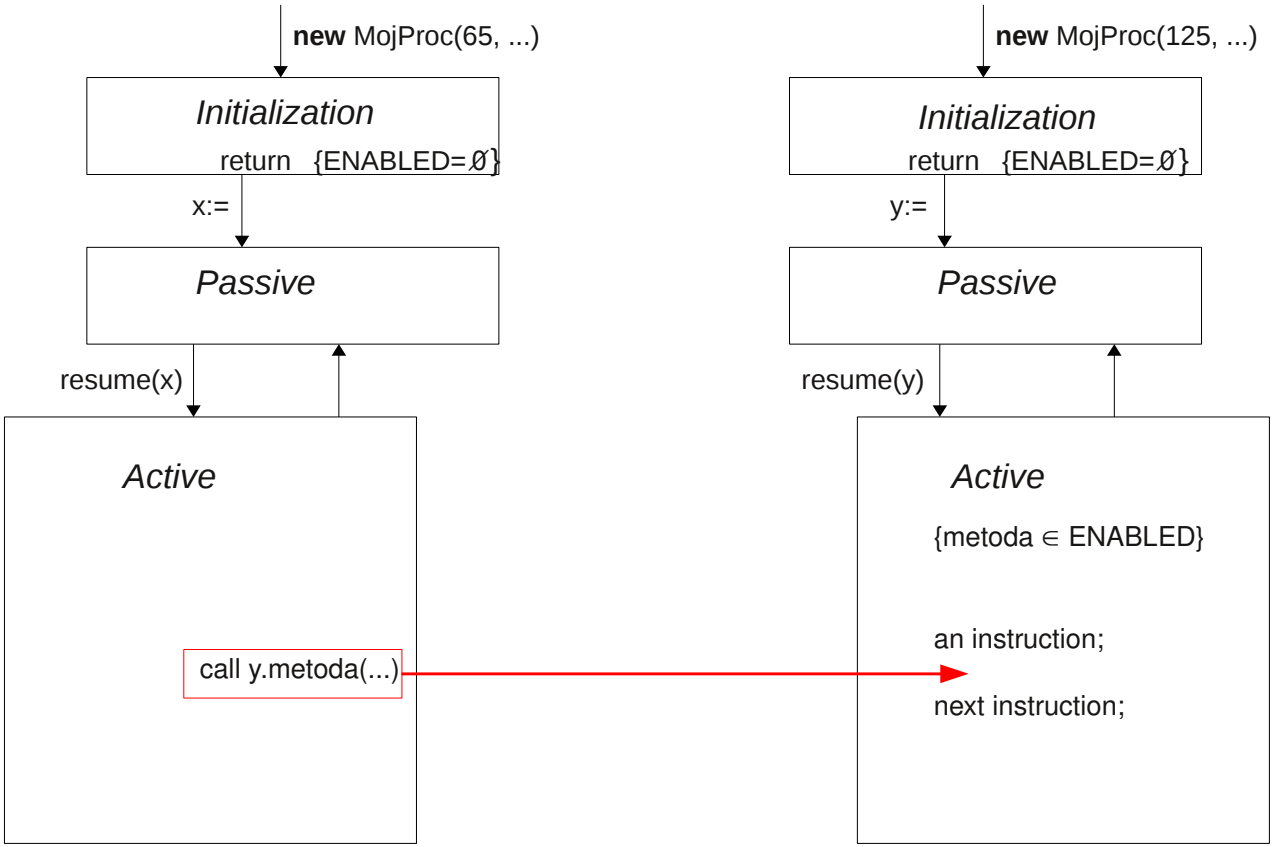
Alien call IV



During execution of instruction accept: ENABLED (of thread y) := ENABLED∪ {metoda}.
While no active object execute instruction „call y.metoda;”, active object y is awaiting.

Rysunek: Protokół alien call przypadek 3

Alien call V



Asynchronous case: when active object x calls metoda in y and metoda ∈ MASK, the thread of active object y is interrupted, object y executes method metoda and returns to its own thread.

Rysunek: Protokół alien call przypadek 4

Alien call VI

Resume

active object x:	active object y
call $y.meth(...)$	object y must exist $\neq$ none & must be ACTIVE & method $meth$ should be enabled, then case A: (asynchronous) object y interrupts its execution, case B: y executes instruction accept: active object y stores its ENABLED, executes method $meth$, passes the **out** results to x, restores ENABLED, resumes its activity.

Każdy moduł może dziedziczyć klasę

W Loglanie'82 każdy rodzaj modułu może dziedziczyć (tj. rozszerzać) klasę.
Możemy więc *wyciągnąć przed nawias* wspólną część kilku algorytmów, zawrzeć ją w klasie C, a później deklarując funkcję f lub procedurę p powiedzieć, że rozszerza ona klasę C.

Przykład

W klasie BST drzew binarnych poszukiwań deklarujemy

Każdy moduł może dziedziczyć klasę

W Loglanie'82 każdy rodzaj modułu może dziedziczyć (tj. rozszerzać) klasę.
Możemy więc *wyciągnąć przed nawias* wspólną część kilku algorytmów, zawrzeć ją w klasie C, a później deklarując funkcję f lub procedurę p powiedzieć, że rozszerza ona klasę C.

Przykład

W klasie BST drzew binarnych poszukiwań deklarujemy
unit *search*: **class**(ee: E, s: Tree); ... **end** search;

Każdy moduł może dziedziczyć klasę

W Loglanie'82 każdy rodzaj modułu może dziedziczyć (tj. rozszerzać) klasę.
Możemy więc *wyciągnąć przed nawias* wspólną część kilku algorytmów, zawrzeć ją w klasie C, a później deklarując funkcję f lub procedurę p powiedzieć, że rozszerza ona klasę C.

Przykład

W klasie BST drzew binarnych poszukiwań deklarujemy
unit *search*: **class**(ee: E, s: Tree); ... **end** search;
unit belongs: *search* **function**(): Boolean; ... **end** belongs;

Każdy moduł może dziedziczyć klasę

W Loglanie'82 każdy rodzaj modułu może dziedziczyć (tj. rozszerzać) klasę.
Możemy więc *wyciągnąć przed nawias* wspólną część kilku algorytmów, zawrzeć ją w klasie C, a później deklarując funkcję f lub procedurę p powiedzieć, że rozszerza ona klasę C.

Przykład

W klasie BST drzew binarnych poszukiwań deklarujemy
unit *search*: **class**(ee: E, s: Tree); ... **end** search;
unit belongs: *search* **function**(): Boolean; ... **end** belongs;
unit insert: *search* **procedure**(); ... **end** insert;

Każdy moduł może dziedziczyć klasę

W Loglanie'82 każdy rodzaj modułu może dziedziczyć (tj. rozszerzać) klasę.
Możemy więc *wyciągnąć przed nawias* wspólną część kilku algorytmów, zawrzeć ją w klasie C, a później deklarując funkcję f lub procedurę p powiedzieć, że rozszerza ona klasę C.

Przykład

W klasie BST drzew binarnych poszukiwań deklarujemy
unit *search*: **class**(ee: E, s: Tree); ... **end** search;
unit belongs: *search* **function**(): Boolean; ... **end** belongs;
unit insert: *search* **procedure**(); ... **end** insert;
unit delete: *search* **procedure**(); ... **end** delete;

Nie ma drugiego języka programowania, który...

miałby następujące cechy:

Nie ma drugiego języka programowania, który...

miałby następujące cechy:

- oferował w pełni obiektowe narzędzie komunikacji pomiędzy procesami-obiektami aktywnymi,

Nie ma drugiego języka programowania, który...

miałby następujące cechy:

- oferował w pełni obiektowe narzędzie komunikacji pomiędzy procesami-obiektami aktywnymi,
- pozwalał w taki sam sposób programować zarówno obliczenia współbieżne jak i obliczenia rozproszone,

Nie ma drugiego języka programowania, który...

miałby następujące cechy:

- oferował w pełni obiektowe narzędzie komunikacji pomiędzy procesami-obiektami aktywnymi,
- pozwalał w taki sam sposób programować zarówno obliczenia współbieżne jak i obliczenia rozproszone,
- pozwalał łączyć maszyny wirtualne w sieciowy wirtualny komputer,

Nie ma drugiego języka programowania, który...

miałby następujące cechy:

- oferował w pełni obiektowe narzędzie komunikacji pomiędzy procesami-obiektami aktywnymi,
- pozwalał w taki sam sposób programować zarówno obliczenia współbieżne jak i obliczenia rozproszone,
- pozwalał łączyć maszyny wirtualne w sieciowy wirtualny komputer,
- umożliwiał bezpieczne usuwanie obiektów,

Nie ma drugiego języka programowania, który...

miałby następujące cechy:

- oferował w pełni obiektowe narzędzie komunikacji pomiędzy procesami-obiektami aktywnymi,
- pozwalał w taki sam sposób programować zarówno obliczenia współbieżne jak i obliczenia rozproszone,
- pozwalał łączyć maszyny wirtualne w sieciowy wirtualny komputer,
- umożliwiał bezpieczne usuwanie obiektów,
- oferował poprawny system współprogramów,

Nie ma drugiego języka programowania, który...

miałby następujące cechy:

- oferował w pełni obiektowe narzędzie komunikacji pomiędzy procesami-obiektami aktywnymi,
- pozwalał w taki sam sposób programować zarówno obliczenia współbieżne jak i obliczenia rozproszone,
- pozwalał łączyć maszyny wirtualne w sieciowy wirtualny komputer,
- umożliwiał bezpieczne usuwanie obiektów,
- oferował poprawny system współprogramów,
- pozwalał zagnieżdżać klasy i dziedziczyć z klas w każdym module,

Nie ma drugiego języka programowania, który...

miałby następujące cechy:

- oferował w pełni obiektowe narzędzie komunikacji pomiędzy procesami-obiektami aktywnymi,
- pozwalał w taki sam sposób programować zarówno obliczenia współbieżne jak i obliczenia rozproszone,
- pozwalał łączyć maszyny wirtualne w sieciowy wirtualny komputer,
- umożliwiał bezpieczne usuwanie obiektów,
- oferował poprawny system współprogramów,
- pozwalał zagnieżdżać klasy i dziedziczyć z klas w każdym module,
- oprócz klas miał moduły process i coroutine,

Nie ma drugiego języka programowania, który...

miałby następujące cechy:

- oferował w pełni obiektowe narzędzie komunikacji pomiędzy procesami-obiektami aktywnymi,
- pozwalał w taki sam sposób programować zarówno obliczenia współbieżne jak i obliczenia rozproszone,
- pozwalał łączyć maszyny wirtualne w sieciowy wirtualny komputer,
- umożliwiał bezpieczne usuwanie obiektów,
- oferował poprawny system współprogramów,
- pozwalał zagnieżdżać klasy i dziedziczyć z klas w każdym module,
- oprócz klas miał moduły process i coroutine,
- i in.

Nie ma drugiego języka programowania, który...

miałby następujące cechy:

- oferował w pełni obiektowe narzędzie komunikacji pomiędzy procesami-obiektami aktywnymi,
- pozwalał w taki sam sposób programować zarówno obliczenia współbieżne jak i obliczenia rozproszone,
- pozwalał łączyć maszyny wirtualne w sieciowy wirtualny komputer,
- umożliwiał bezpieczne usuwanie obiektów,
- oferował poprawny system współprogramów,
- pozwalał zagnieżdżać klasy i dziedziczyć z klas w każdym module,
- oprócz klas miał moduły process i coroutine,
- i in.

Jakiś podobny język powstanie za parę lat, w USA, Chinach lub gdzie indziej.

Nie ma drugiego języka programowania, który...

miałby następujące cechy:

- oferował w pełni obiektowe narzędzie komunikacji pomiędzy procesami-obiektami aktywnymi,
- pozwalał w taki sam sposób programować zarówno obliczenia współbieżne jak i obliczenia rozproszone,
- pozwalał łączyć maszyny wirtualne w sieciowy wirtualny komputer,
- umożliwiał bezpieczne usuwanie obiektów,
- oferował poprawny system współprogramów,
- pozwalał zagnieżdżać klasy i dziedziczyć z klas w każdym module,
- oprócz klas miał moduły process i coroutine,
- i in.

Jakiś podobny język powstanie za parę lat, w USA, Chinach lub gdzie indziej.

Loglan'82 uzyskał to wszystko w **1982**, dzięki *wsadowi* badań naukowych.

Plan prezentacji

Features	Simula67	obj.Pascal	C++	Modula3	Smalltalk	Eiffel	Ada	Beta	Java	**Loglan82**
Modularisation										
nesting of modules	+	+	–	–	–	–	+	+	+	+
inheritance	+	+	+	+	+	+	–	+	+	+
– multilevel	–	–	–	–	–	–	–	+	+	+
– multiple	–	–	+	–	+	+	–	–	+	–
– in functions	–	–	–	–	–	–	–	+	–	+
static binding	+	+	–	+	–	–	+	+	+	+
Classes & Objects	+	+	+	+	+	+	+	+	+	+
Coroutines	+	–	–	+	–	–	–	+	–	+
Processes	–	–	–	+	–	–	+	+	+	+
– alien calls	–	–	–	–	–	–	–	–	–	+
Signals & Exceptions	–	–	+	–	–	–	+	+	+	+
Safety										
safe dealocation	–	–	–	–	–	–	–	–	–	+
type checking	+	+	–	–	–	–	+	+	+	+
protection of private	+	–	–	+	–	–	+	+	+	+
Genericity										
types as formal parameters	–	–	–	–	–	–	–	–	–	+
virtual methods	+		+		+	+	–	+	+	+
overloading	–		+	+		+	+	+	+	–

Porównanie języków programowania obiektowego I

Features	Simula 67	obj.Pascal	C++	Modula 3	Smalltalk	Eiffel	Ada	Beta	Java	**Loglan 82**
Modularisation										
nesting of modules	+	+	–	–	–	–	+	+	+	+
inheritance	+	+	+	+	+	+	–	+	+	+
– multilevel	–	–	–	–	–	–	–	+	+	+
– multiple	–	–	+	–	+	+	–	–	+	–
– in functions	–	–	–	–	–	–	–	+	–	+
static binding	+	+	–	+	–	–	+	+	+	+

Porównanie języków programowania obiektowego II

Features	Simula 67	obj.Pascal	C++	Modula 3	Smalltalk	Eiffel	Ada	Beta	Java	**Loglan 82**
Classes & Objects	+	+	+	+	+	+	+	+	+	+
Coroutines	+	–	–	+	–	–	–	+	–	+
Processes	–	–	–	+	–	–	+	+	+	+
– alien calls	–	–	–	–	–	–	–	–	–	+
Signals & Exceptions	–	–	–	–	–	–	+	+	+	+

Porównanie języków programowania obiektowego III

Features	Simula67	obj.Pascal	C++	Modula3	Smalltalk	Eiffel	Ada	Beta	Java	**Loglan82**
Safety										
safe dealocation	−	−	−	−	−	−	−	−	−	+
type checking	+	+	−	−	−	−	+	+	+	+
protection of private	+	−	−	+	−	−	+	+	+	+
Genericity										
types as formal parameters	−	−	−	−	−	−	−	−	−	+
virtual methods	+		+		+	+	−	+	+	+
overloading	−		+	+		+	+	+	+	−

Jutro

Problemy i zadania

Problemy I

Problemy I

- Czy można rozszerzyć protokół alien call na programowanie równoległe?

Problemy I

- Czy można rozszerzyć protokół alien call na programowanie równoległe?
- Czy przekazywanie nazwy klasy jako parametru formalnego stworzy nowe możliwości?

Problemy I

- Czy można rozszerzyć protokół alien call na programowanie równoległe?
- Czy przekazywanie nazwy klasy jako parametru formalnego stworzy nowe możliwości?
- Porównać alien call z pojęciem 'chord' w Polyphony - rozszerzeniu języka C#.

Problemy I

- Czy można rozszerzyć protokół alien call na programowanie równoległe?
- Czy przekazywanie nazwy klasy jako parametru formalnego stworzy nowe możliwości?
- Porównać alien call z pojęciem 'chord' w Polyphony - rozszerzeniu języka C#.
- Zbudować wzorcową formalną semantykę Loglanu.

Problemy I

- Czy można rozszerzyć protokół alien call na programowanie równoległe?
- Czy przekazywanie nazwy klasy jako parametru formalnego stworzy nowe możliwości?
- Porównać alien call z pojęciem 'chord' w Polyphony - rozszerzeniu języka C#.
- Zbudować wzorcową formalną semantykę Loglanu.
- Czy operacja $attach(x)$ na współprogramach jest cyklicznym obrotem stosu rekordów aktywacji takim, że na wierzchołku stosu znajduje się łańcuch dynamiczny współprogramu x?

Problemy II

Problem o zupełnie innym wymiarze.

Nowy Loglan stanie się częścią ogromnego projektu SpecVer - środowiska dla pracy nad specyfikacja oprogramowania, nad oprogramowaniem i nad weryfikacja oprogramowania względem specyfikacji. W środowisku SpecVer potrzebne będa narzędzia wspomagajace pracę twórców specyfikacji.
Potrzebne będa narzędzia do pracy nad oprogramowaniem,a więc kompilatory, ale także narzędzia do eksperymentowania z programami i graficznej wizualizacji, np graficzny debugger
Narzędzia dla audytorów, czyli specjalistów badających czy oprogramowanie P jest zgodne ze specyfikacją S będą dopiero powstawać. Ta praca nie da się zautomatyzować - i bardzo dobrze. Ale pomocą mogą być proof checkery, provery i specjalistyczne edytory..

Zadania

Zadania

- Ponowna kompilacja kompilatora Loglanu.

Zadania

- Ponowna kompilacja kompilatora Loglanu.
- Przepisanie środowiska VLP na nowo tym razem w qt4.

Zadania

- Ponowna kompilacja kompilatora Loglanu.
- Przepisanie środowiska VLP na nowo tym razem w qt4.
- Przeniesienie środowiska VLP na platformę Windows.

Zadania

- Ponowna kompilacja kompilatora Loglanu.
- Przepisanie środowiska VLP na nowo tym razem w qt4.
- Przeniesienie środowiska VLP na platformę Windows.
- **Nowa wersja języka Loglan** – LEM'12.

Zadania

- Ponowna kompilacja kompilatora Loglanu.
- Przepisanie środowiska VLP na nowo tym razem w qt4.
- Przeniesienie środowiska VLP na platformę Windows.
- **Nowa wersja języka Loglan** – LEM'12.
- Rozwijanie zastosowań Loglanu.

Zadania

- Ponowna kompilacja kompilatora Loglanu.
- Przepisanie środowiska VLP na nowo tym razem w qt4.
- Przeniesienie środowiska VLP na platformę Windows.
- **Nowa wersja języka Loglan** – LEM'12.
- Rozwijanie zastosowań Loglanu.
- VLP czy Eclipse?

Zadania

- Ponowna kompilacja kompilatora Loglanu.
- Przepisanie środowiska VLP na nowo tym razem w qt4.
- Przeniesienie środowiska VLP na platformę Windows.
- **Nowa wersja języka Loglan** – LEM'12.
- Rozwijanie zastosowań Loglanu.
- VLP czy Eclipse?
- Biblioteka klas.

Zadania

- Ponowna kompilacja kompilatora Loglanu.
- Przepisanie środowiska VLP na nowo tym razem w qt4.
- Przeniesienie środowiska VLP na platformę Windows.
- **Nowa wersja języka Loglan** – LEM'12.
- Rozwijanie zastosowań Loglanu.
- VLP czy Eclipse?
- Biblioteka klas.
- Porównać Java RMI i alien call.

Zadania

- Ponowna kompilacja kompilatora Loglanu.
- Przepisanie środowiska VLP na nowo tym razem w qt4.
- Przeniesienie środowiska VLP na platformę Windows.
- **Nowa wersja języka Loglan** – LEM'12.
- Rozwijanie zastosowań Loglanu.
- VLP czy Eclipse?
- Biblioteka klas.
- Porównać Java RMI i alien call.
- Zrealizować wtyczkę loglanowską do przeglądarki.

Zadania

- Ponowna kompilacja kompilatora Loglanu.
- Przepisanie środowiska VLP na nowo tym razem w qt4.
- Przeniesienie środowiska VLP na platformę Windows.
- **Nowa wersja języka Loglan** – LEM'12.
- Rozwijanie zastosowań Loglanu.
- VLP czy Eclipse?
- Biblioteka klas.
- Porównać Java RMI i alien call.
- Zrealizować wtyczkę loglanowską do przeglądarki.
- Napisać maszynę wirtualna LEM12 w Loglanie

Zadania II – Przekazywać doświadczenie Loglanu'82 innym

Jedną rzecz już zrobiliśmy: Korzystając z doświadczenia z Loglanem'82,

- sformułowaliśmy problem wyznaczania bezposrednich superklas w Javie,
- podaliśmy algorytm (w dwu wersjach)
- i wykazaliśmy jego poprawnośc i zupełność [LSW08, LSW09].

Pozostaje sporo do zrobienia:

- Czy protokół alien call jest implementowalny w Javie?
- Czy protokól alien call jest implementowalny w C++?
- Zaimplementować system Kreczmara w C++?
- Zaimplementować współprogramy w C++.
- i in.

Pytanie Zasadnicze

Czy można zdefiniować nowy język programowania LEM'12 o następujących cechach:

Pytanie Zasadnicze

Czy można zdefiniować nowy język programowania LEM'12 o następujących cechach:

- nowy język zachowa wszystkie osiągnięcia Loglanu'82,

Pytanie Zasadnicze

Czy można zdefiniować nowy język programowania LEM'12 o następujących cechach:

- nowy język zachowa wszystkie osiągnięcia Loglanu'82,
- wykorzysta co tylko się da z dorobku Javy, C#, i innych nowszych języków programowania,

Pytanie Zasadnicze

Czy można zdefiniować nowy język programowania LEM'12 o następujących cechach:

- nowy język zachowa wszystkie osiągnięcia Loglanu'82,
- wykorzysta co tylko się da z dorobku Javy, C#, i innych nowszych języków programowania,
- w miejsce modułów `interface` wprowadzi moduły `specification`, w celu stworzenia środowiska SpecVer. (To środowisko ma stworzyć wspólne ramy i wspomagać prace nad specyfikacją, implementacją i weryfikacją oprogramowania.)

Pytanie Zasadnicze

Czy można zdefiniować nowy język programowania LEM'12 o następujących cechach:

- nowy język zachowa wszystkie osiągnięcia Loglanu'82,
- wykorzysta co tylko się da z dorobku Javy, C#, i innych nowszych języków programowania,
- w miejsce modułów `interface` wprowadzi moduły `specification`, w celu stworzenia środowiska SpecVer. (To środowisko ma stworzyć wspólne ramy i wspomagać prace nad specyfikacją, implementacją i weryfikacją oprogramowania.)
- Język umożliwi programowanie równoległe wykorzystujące wszystkie procesory (lub rdzenie) komputera.

Pytanie Zasadnicze

Czy można zdefiniować nowy język programowania LEM'12 o następujących cechach:

- nowy język zachowa wszystkie osiągnięcia Loglanu'82,
- wykorzysta co tylko się da z dorobku Javy, C#, i innych nowszych języków programowania,
- w miejsce modułów `interface` wprowadzi moduły `specification`, w celu stworzenia środowiska SpecVer. (To środowisko ma stworzyć wspólne ramy i wspomagać prace nad specyfikacją, implementacją i weryfikacją oprogramowania.)
- Język umożliwi programowanie równoległe wykorzystujące wszystkie procesory (lub rdzenie) komputera.
- i in.

Ile to jest warte?

- wartości poznawcze
  nie wiem ile wyników zostanie uzyskanych i opublikowanych?
  ile doktoratów, habilitacji można będzie uznać za owoc pracy w nowym projekcie?
- wartości materialne
  chyba nie mniej niż Java?!
  a to mogą być kwoty pokaźne.

- Zapraszam wszystkich chętnych do współpracy.
- Naszą i Twoją, szansą jest praca zespołowa.

Kontakt

Napisz:
salwicki@mimuw.edu.pl

Wersja do czytania:
http://duch.mimuw.edu.pl/∼salwicki/loglan/Doc/Zaproszenie-do-wspolpracy.pdf

Ta prezentacja:
http://duch.mimuw.edu.pl/∼salwicki/loglan/Doc/seminariumPoznan.pdf

Dziękuję za uwagę.