Weryfikacja modelowa
Język Alvis

Marcin Szpyrka

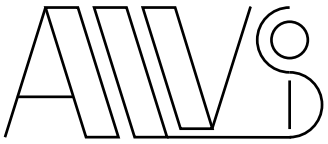
Katedra Informatyki Stosowanej
AGH w Krakowie

2014/15

Literatura

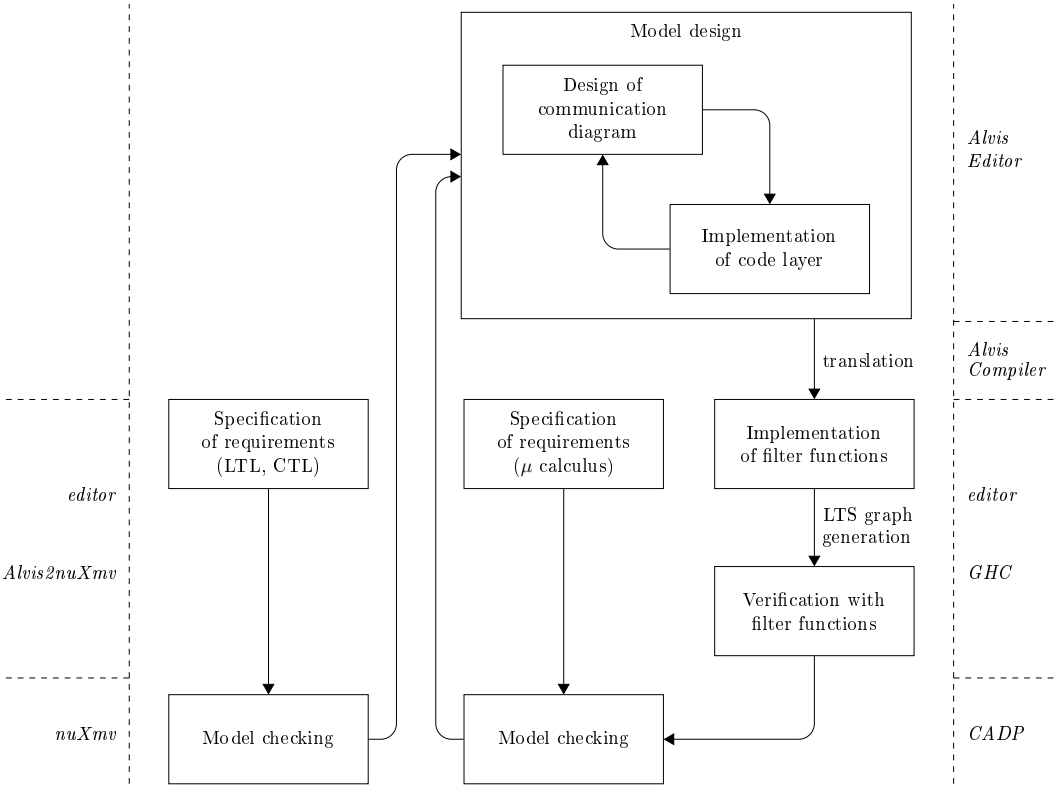
1. Marcin Szpyrka: Modelowanie systemów współbieżnych w języku Alvis. Wydawnictwa AGH, Kraków, 2013.
2. Alvis Manual, `http://fm.kis.agh.edu.pl`

Alvis



- Język Alvis powstał jako wynik poszukiwania języka modelowania, który z jednej strony byłby łatwy do opanowania przez inżyniera informatyka, a z drugiej pozwalał na formalną weryfikację wytworzonego modelu.
- Język Alvis jest rozwijany w Katedrze Informatyki Stosowanej AGH.
- Pomimo inspiracji algebrami procesów oraz wstępnych zamierzeń, co wyraża m.in. nazwa Alvis = Algebra + visual, język Alvis nie jest algebrą procesów. Różne elementy języka są wynikiem doświadczeń z wieloma formalizmami i językami, które stosuje się do modelowania lub implementacji systemów współbieżnych i/lub wbudowanych.
- Jednak, wiele terminów stosowanych w języku Alvis (agent, port, kanał komunikacyjny) zapożyczono z algebry procesów CCS.
- Duży wpływ na niektóre koncepcje języka Alvis miał język Ada – agenty aktywne i pasywne, instrukcja **`select`**, komunikacja.

Proces modelowania i weryfikacji systemów w języku Alvis

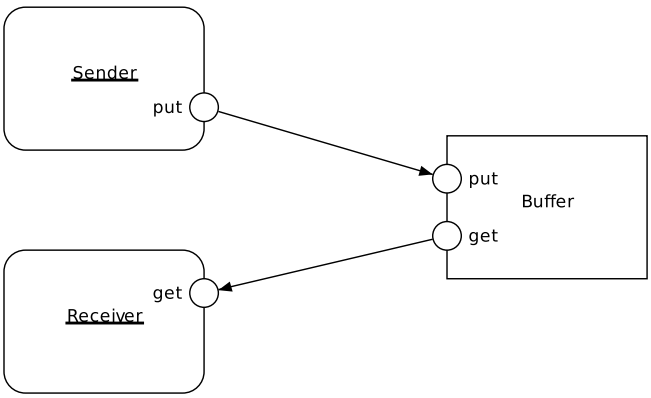


Warstwy modelu

- Diagram komunikacji stanowi warstwę graficzną modelu. Służy do definiowania połączeń komunikacyjnych między agentami.
  Diagram komunikacji jest konstrukcją hierarchiczną, w której pojedyncze agenty mogą być zastępowane poddiagramami, które szczegółowo opisują dany fragment modelowanego systemu.
- Warstwa kodu służy do: definiowania typów danych dla parametrów, definiowania funkcji do manipulowania wartościami parametrów, specyfikowania zachowania się otoczenia i opisywania dynamiki poszczególnych agentów.
- Warstwa systemowa decyduje o semantyce modelu. W zależności od wybranej warstwy systemowej konstruujemy model dla systemów jedno- lub wieloprocesorowych.

UWAGA: Z punktu widzenia projektanta konieczne jest przygotowanie tylko diagramu komunikacji i warstwy kodu.

Alvis – przykład




```
agent Sender {
  loop {
    out put;
  }
}

agent Buffer {
  i :: Int = 0;
  proc (i == 0) put {
    in put;
    i = 1;
  }
  proc (i /= 0) get {
    out get;
    i = 0;
  }
}

agent Receiver {
  loop {
    in get;
  }
}
```

Podstawowe pojęcia (1)

- Termin agent odnosi się w Alvisie do dowolnego wyróżnionego fragmentu systemu, który ma własną tożsamość trwającą w czasie i któremu można przypisać jego stan.
- W Alvisie wyróżnia się dwa rodzaje agentów. Agenty aktywne są podobne do zadań w języku Ada – mają własny wątek obliczeniowy, zaś agenty pasywne, służą do reprezentowania współdzielonych zasobów i zapewniają wzajemne wykluczanie w dostępie do tych zasobów.
- Agenty są reprezentowane za pomocą prostokątów, przy czym w przypadku agentów aktywnych używane są prostokąty z zaokrąglonymi rogami.
- Nazwa agenta jest jego identyfikatorem i musi zaczynać się od wielkiej litery (poza tym może zawierać litery alfabetu łacińskiego, cyfry i znaki podkreślenia). Nazwy agentów w modelu nie mogą się powtarzać.
- Agent może komunikować się z innymi agentami lub swoim otoczeniem poprzez porty. Są one reprezentowane przez okręgi umieszczane na krawędziach odpowiednich prostokątów. Etykiety portów muszą rozpoczynać się od małej litery. Nazwy portów mogą się powtarzać w modelu, ale jeden agent nie może zawierać dwóch portów o identycznych nazwach. Każdy z portów można jednoznacznie wskazać, podając nazwę agenta i nazwę portu, np. $X.p$.

Podstawowe pojęcia (2)

- Dwa porty należące do różnych agentów można połączyć ze sobą, co pozwala na przesyłanie informacji między agentami.
- W przypadku połączeń jednokierunkowych grot wskazuje port wejściowy dla danego połączenia, tj. port do którego przesyłane są sygnały (informacje).
- Połączenie dwukierunkowe (bez grotu) reprezentuje w rzeczywistości dwa połączenia o przeciwnych kierunkach przesyłania danych.
- Agent aktywny może być uruchomiony bezpośrednio po starcie systemu lub wzbudzony później przez innego agenta. Agenty aktywne, które są uruchamiane przy starcie systemu, mają podkreślone nazwy. Wymagane jest, aby co najmniej jeden agent był aktywny przy starcie, w przeciwnym razie system nie będzie mógł wykonać żadnej akcji.
- Diagram komunikacji stanowi jedną z trzech warstw systemu. Pokazuje strukturę połączeń ze względu na komunikację między agentami, ale nie zawiera informacji o sposobie funkcjonowania poszczególnych agentów. Dynamika agentów jest opisywana w tzw. warstwie kodu.
- Użycie warstwy systemowej α^0 oznacza, że każdy agent aktywny ma dostęp do własnego procesora i może realizować swoje instrukcje równolegle z innymi agentami. Aktualnie obsługiwana jest tylko warstwa α^0.

Alvis – model

Modelem w języku Alvis nazywamy trójkę $\mathbf{A} = (H, B, \varphi)$, gdzie:

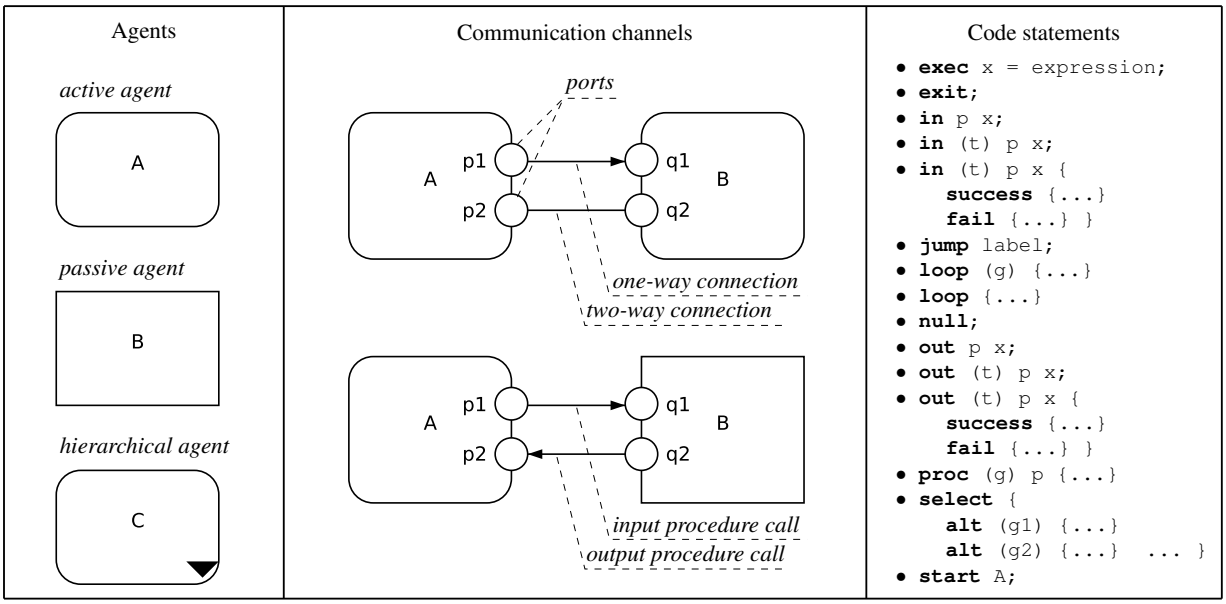
- $H = (\mathcal{D}, \gamma)$ jest hierarchicznym diagramem komunikacji,
- B jest syntaktycznie poprawną warstwą kodu,
- φ jest warstwą systemową.

Ponadto każdy niehierarchiczny agent X należący do diagramu H musi być zdefiniowany w warstwie kodu B i na odwrót – każdy agent zdefiniowany w warstwie kodu musi należeć do diagramu H.

UWAGA: Termin syntaktycznie poprawna warstwa kodu oznacza nie tylko poprawność składniową użytych instrukcji, ale również poprawne użycie portów, np. tylko porty wejściowe mogą być użyte jako argumenty instrukcji **`in`** i tylko porty wyjściowe mogą być użyte jako argumenty instrukcji **`out`**. Dotyczy to również kolejności użycia instrukcji **`in`** i **`out`** wewnątrz procedur, gdy argumentami są porty proceduralne. Instrukcje takie muszą być wykonane przed zakończeniem procedury.

Rozważamy wyłącznie warstwę systemową oznaczoną symbolem α^0, której użycie oznacza, że każdy z aktywnych agentów ma dostęp do własnego procesora, a w sytuacjach konfliktowych o pierwszeństwie decydują priorytety agentów. Jeżeli dwa lub więcej agentów o tym samym priorytecie rywalizuje o te same zasoby, to system zachowuje się niedeterministycznie.

Elementy języka Alvis



Porty

- $\mathcal{P}(X)$ – zbiór portów agenta X;
- $\mathcal{P}_{in}(X)$ – zbiór portów wejściowych agenta X, tj. portów z co najmniej jednym połączeniem jednokierunkowym, prowadzącym do tego portu lub z co najmniej jednym połączeniem dwukierunkowym;
- $\mathcal{P}_{out}(X)$ – zbiór portów wyjściowych tj. portów z co najmniej jednym połączeniem jednokierunkowym, prowadzącym od tego portu lub z co najmniej jednym połączeniem dwukierunkowym;
- $\mathcal{P}_{unc}(X) = \mathcal{P}(X) \setminus (\mathcal{P}_{in}(X) \cup \mathcal{P}_{out}(X))$ – zbiór portów izolowanych;
- $\mathcal{P}_{proc}(X)$ oznacza zbiór portów proceduralnych, tj. portów agenta pasywnego ze zdefiniowaną procedurą (nazwa takiego portu jest traktowana jako nazwa procedury);
- Dla zbioru agentów W: $\mathcal{P}(W) = \bigcup_{X \in W} \mathcal{P}(X)$, $\mathcal{P}_{in}(W) = \bigcup_{X \in W} \mathcal{P}_{in}(X)$ itd.
- $\mathcal{P}$ – zbiór wszystkich portów w modelu, $\mathcal{P}_{in}$ – zbiór wszystkich portów wejściowych itd.
- $\mathcal{N}(Y)$ – zbiór nazw portów należących do Y, np. jeżeli diagram zawiera wyłącznie agenta X_1 z jednym portem p i agenta X_2 również tylko z jednym portem p, to $\mathcal{P} = \{X_1.p, X_2.p\}$, zaś $\mathcal{N}(\mathcal{P}) = \{p\}$.

Niehierarchiczny diagram komunikacji

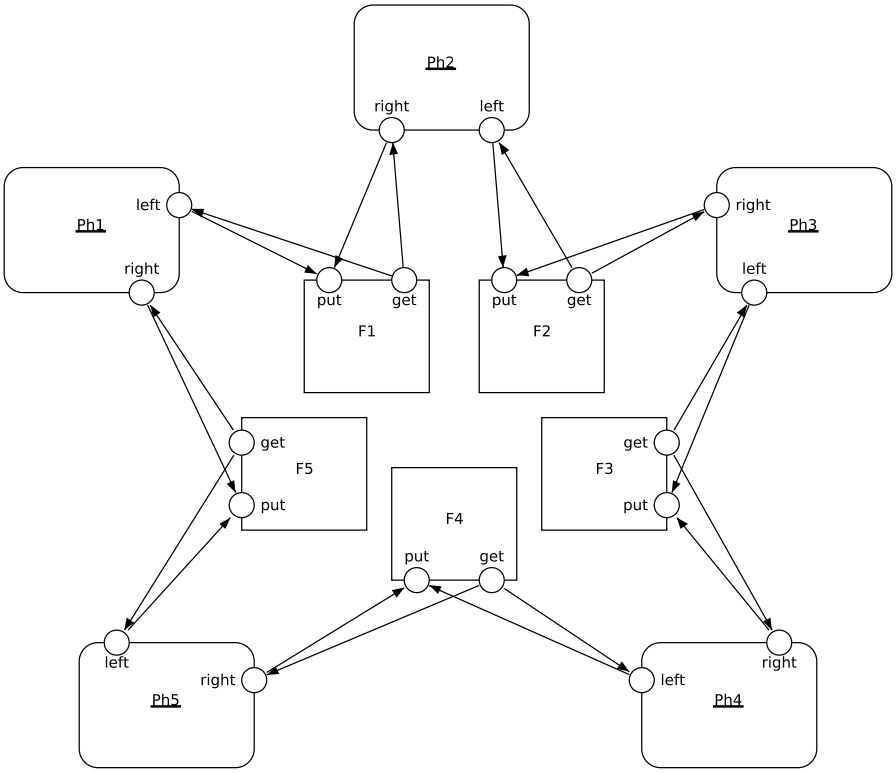
Niehierarchicznym diagramem komunikacji nazywamy trójkę $D = (\mathcal{A}, \mathcal{C}, \sigma)$, gdzie:

- $\mathcal{A} = \{X_1, \ldots, X_n\}$ jest zbiorem agentów, składającym się z dwóch rozłącznych zbiorów agentów aktywnych $\mathcal{A}_A$ i pasywnych $\mathcal{A}_P$ takich, że $\mathcal{A} = \mathcal{A}_A \cup \mathcal{A}_P$;
- $\mathcal{C} \subseteq \mathcal{P} \times \mathcal{P}$ jest relacją komunikacji taką, że:

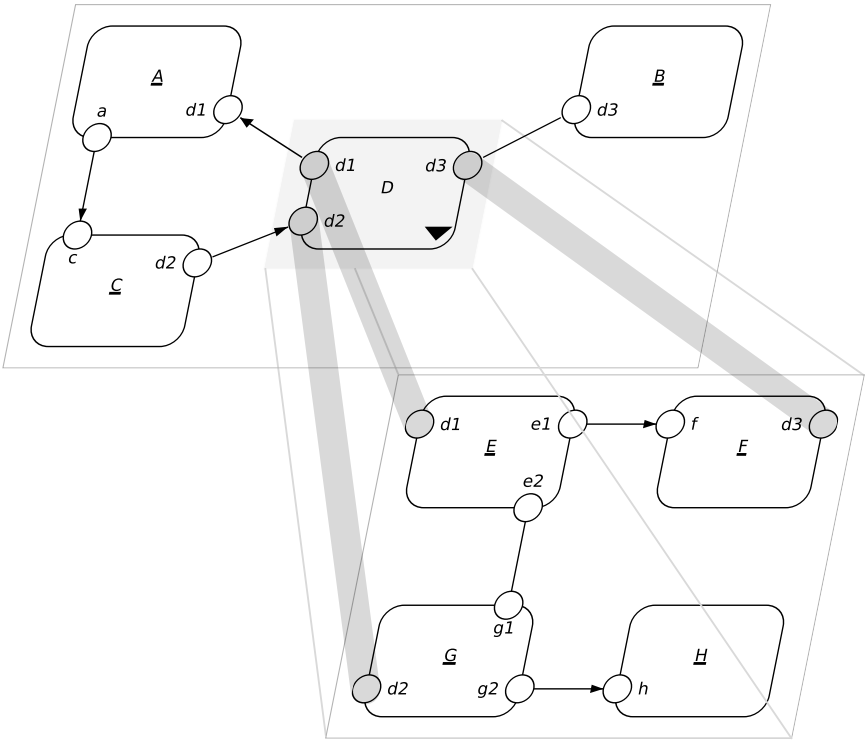
$$\forall_{X \in \mathcal{A}} \ (\mathcal{P}(X) \times \mathcal{P}(X)) \cap \mathcal{C} = \emptyset \tag{1}$$
$$\mathcal{P}_{proc} \cap \mathcal{P}_{in} \cap \mathcal{P}_{out} = \emptyset \tag{2}$$
$$(p,q) \in (\mathcal{P}(\mathcal{A}_A) \times \mathcal{P}(\mathcal{A}_P)) \cap \mathcal{C} \Rightarrow q \in \mathcal{P}_{proc} \tag{3}$$
$$(p,q) \in (\mathcal{P}(\mathcal{A}_P) \times \mathcal{P}(\mathcal{A}_A)) \cap \mathcal{C} \Rightarrow p \in \mathcal{P}_{proc} \tag{4}$$
$$(p,q) \in (\mathcal{P}(\mathcal{A}_P) \times \mathcal{P}(\mathcal{A}_P)) \cap \mathcal{C} \Rightarrow (p \in \mathcal{P}_{proc} \wedge q \notin \mathcal{P}_{proc}) \vee$$
$$\vee \ (q \in \mathcal{P}_{proc} \wedge p \notin \mathcal{P}_{proc}) \tag{5}$$

- $\sigma \colon \mathcal{A}_A \to \{False, True\}$ jest funkcją aktywności, która wskazuje uruchamiane przy starcie agenty aktywne.

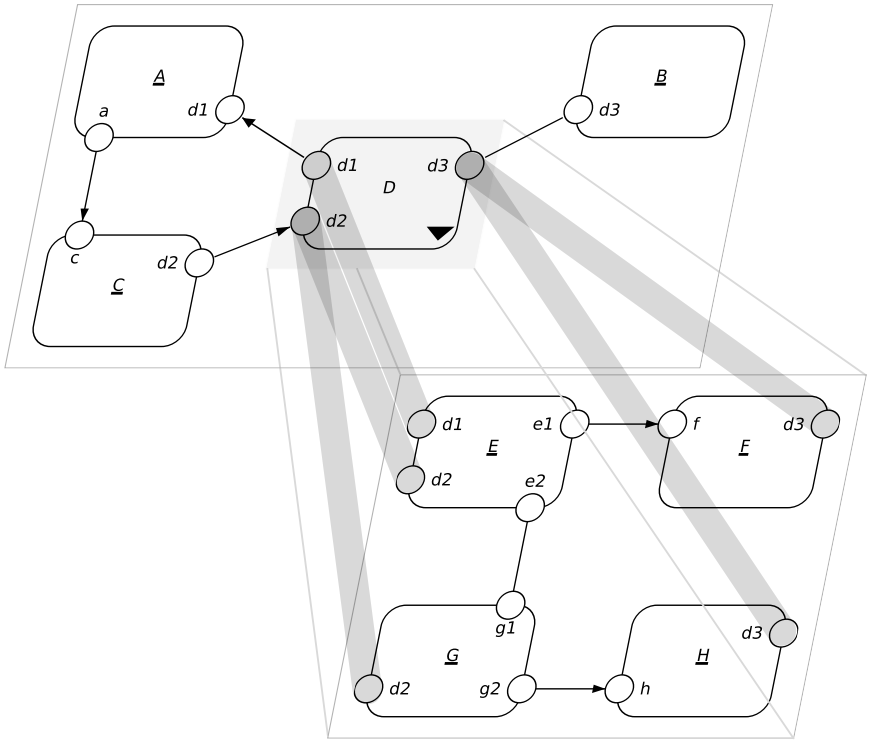
Diagram komunikacji dla problemu ucztujących filozofów



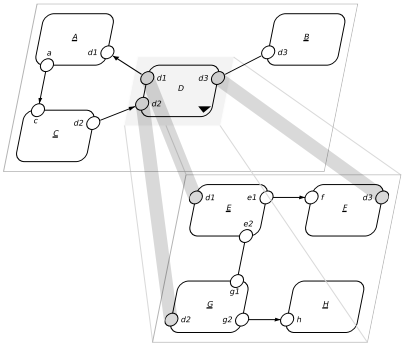
Hierarchia – podstawienie proste

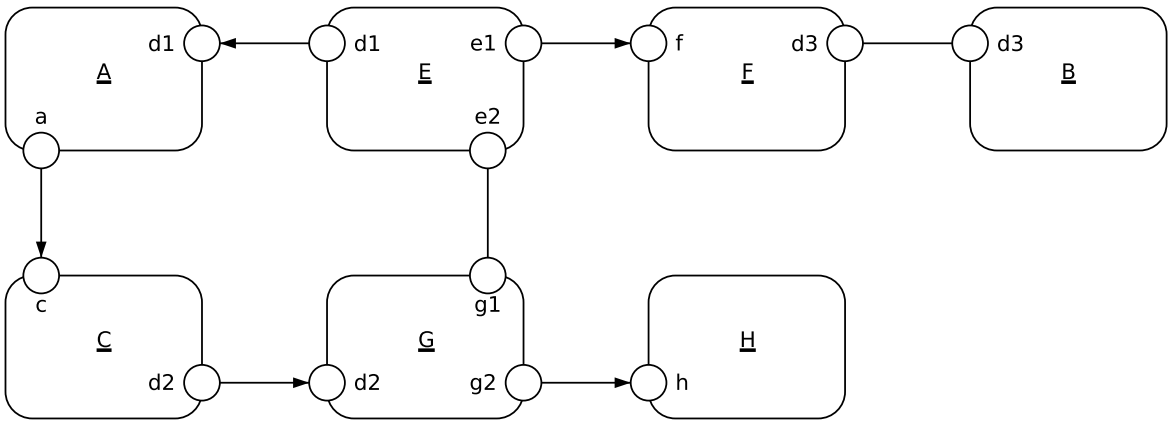


Hierarchia – podstawienie uogólnione

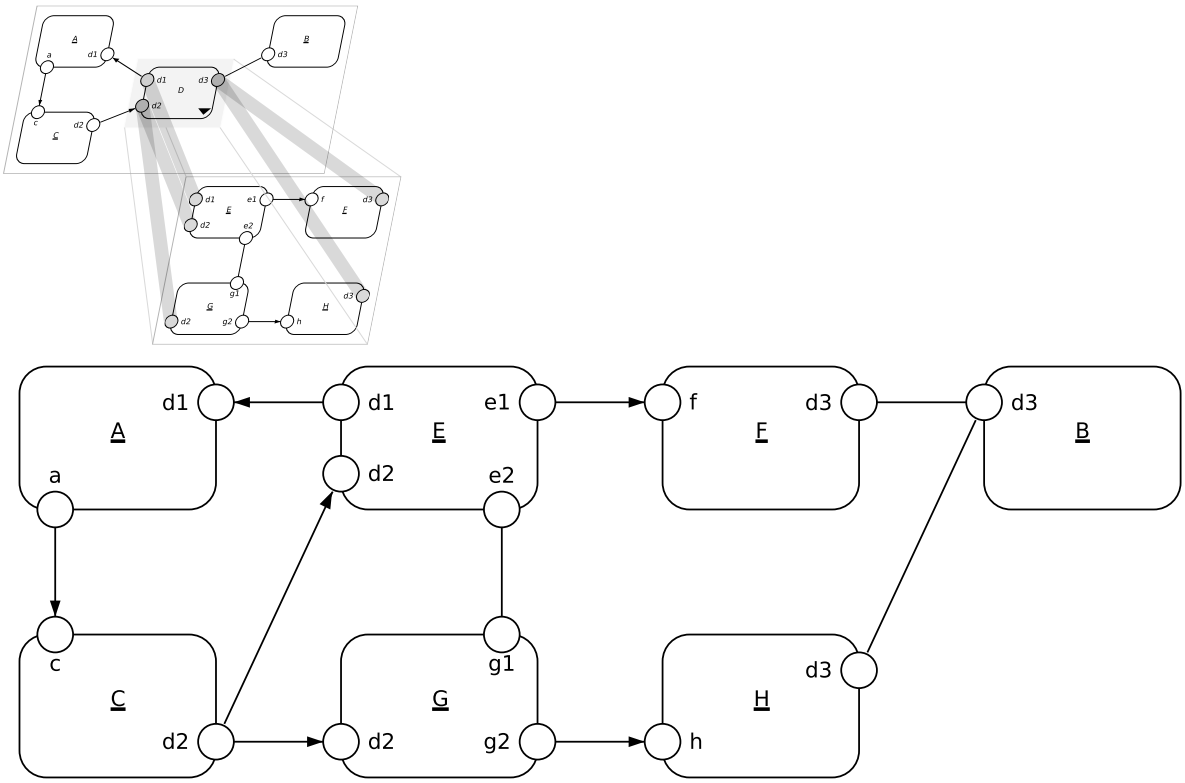


Wynik operacji analizy – podstawienie proste





Wynik operacji analizy – podstawienie uogólnione



Hierarchiczne diagramy komunikacji – definicja strony

Stroną nazywamy trójkę $D^i = (\mathcal{A}^i, \mathcal{C}^i, \sigma^i)$, gdzie:

- $\mathcal{A}^i = \{X_1^i, \ldots, X_n^i\}$ jest zbiorem agentów, z wyróżnionymi podzbiorami $\mathcal{A}_A^i$ agentów aktywnych, $\mathcal{A}_P^i$ agentów pasywnych i $\mathcal{A}_H^i$ agentów hierarchicznych takimi, że $\mathcal{A}^i = \mathcal{A}_A^i \cup \mathcal{A}_P^i \cup \mathcal{A}_H^i$ i $\mathcal{A}_A^i$, $\mathcal{A}_P^i$, $\mathcal{A}_H^i$ są parami rozłączne;
- $\mathcal{C}^i \subseteq \mathcal{P}^i \times \mathcal{P}^i$, gdzie $\mathcal{P}^i = \bigcup_{X \in \mathcal{A}^i} \mathcal{P}(X)$, jest relacją komunikacji taką, że:

$$\forall_{X \in \mathcal{A}^i} \ (\mathcal{P}(X) \times \mathcal{P}(X)) \cap \mathcal{C}^i = \emptyset \tag{6}$$

$$\mathcal{P}_{proc}^i \cap \mathcal{P}_{in}^i \cap \mathcal{P}_{out}^i = \emptyset \tag{7}$$

$$\mathcal{P}_{proc}^i \cap \mathcal{P}(\mathcal{A}_H^i) = \emptyset \tag{8}$$

$$(p,q) \in (\mathcal{P}(\mathcal{A}_A^i) \times \mathcal{P}(\mathcal{A}_P^i)) \cap \mathcal{C}^i \Rightarrow q \in \mathcal{P}_{proc}^i \tag{9}$$

$$(p,q) \in (\mathcal{P}(\mathcal{A}_P^i) \times \mathcal{P}(\mathcal{A}_A^i)) \cap \mathcal{C}^i \Rightarrow p \in \mathcal{P}_{proc}^i \tag{10}$$

$$(p,q) \in (\mathcal{P}(\mathcal{A}_P^i) \times \mathcal{P}(\mathcal{A}_P^i)) \cap \mathcal{C}^i \Rightarrow (p \in \mathcal{P}_{proc}^i \wedge q \notin \mathcal{P}_{proc}^i) \vee$$
$$\vee \ (q \in \mathcal{P}_{proc}^i \wedge p \notin \mathcal{P}_{proc}^i) \tag{11}$$

$$(p,q) \in (\mathcal{P}(\mathcal{A}_P^i) \times \mathcal{P}(\mathcal{A}_H^i)) \cap \mathcal{C}^i \Rightarrow (q,p) \notin \mathcal{C}^i \tag{12}$$

$$(p,q) \in (\mathcal{P}(\mathcal{A}_H^i) \times \mathcal{P}(\mathcal{A}_P^i)) \cap \mathcal{C}^i \Rightarrow (q,p) \notin \mathcal{C}^i \tag{13}$$

- $\sigma^i \colon \mathcal{A}_A^i \to \{False, True\}$ jest funkcją aktywności.

Hierarchiczny diagram komunikacji – definicja

Grafem skierowanym nazywamy trójkę $\mathcal{G} = (V, E, L)$, gdzie:

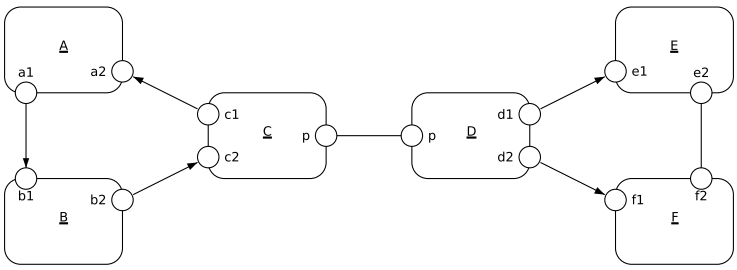
- V jest zbiorem węzłów (wierzchołków) grafu;
- L jest zbiorem etykiet krawędzi (łuków) grafu;
- $E \subseteq V \times L \times V$ jest zbiorem krawędzi (łuków) grafu.

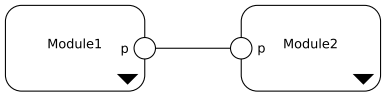
Hierarchicznym diagramem komunikacji nazywamy taką parę $H = (\mathcal{D}, \gamma)$, gdzie $\mathcal{D} = \{D^1, \ldots, D^k\}$ jest zbiorem stron, takim że zbiory agentów $\mathcal{A}^i$ $(i = 1, \ldots, k)$ są parami rozłączne i $\gamma \colon \mathcal{A}_H \to \mathcal{D}$, gdzie $\mathcal{A}_H = \bigcup_{i=1,\ldots,k} \mathcal{A}^i_H$, jest funkcją podstawień, taką że:

- γ jest iniekcją;
- dla dowolnego $X \in \mathcal{A}_H$, X i $\gamma(X)$ spełniają warunki podstawienia prostego lub uogólnionego;
- graf skierowany $\mathcal{G} = (\mathcal{D}, E, \mathcal{A}_H)$, gdzie $(D^i, X^i_k, D^j) \in E$ wtw, gdy $X^i_k \in \mathcal{A}^i_H$ i $\gamma(X^i_k) = D^j$ jest drzewem lub lasem.

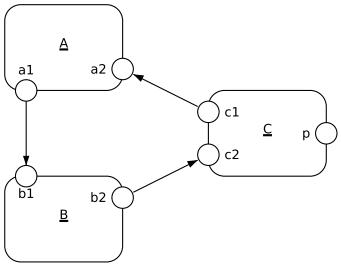
Graf skierowany $\mathcal{G} = (\mathcal{D}, E, \mathcal{A}_H)$ nazywamy grafem hierarchii stron. Węzły w takim grafie reprezentują strony diagramu, zaś krawędzie etykietowane nazwami agentów hierarchicznych reprezentują funkcję podstawień γ. Strony należące do zbioru $\mathcal{D} \setminus \gamma(\mathcal{A}_H)$ nazywamy stronami głównymi diagramu.

Użycie hierarchii – moduły

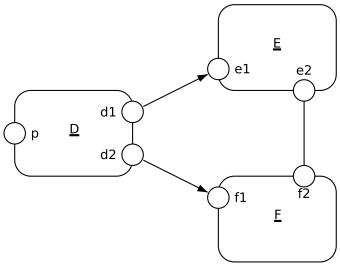




a)

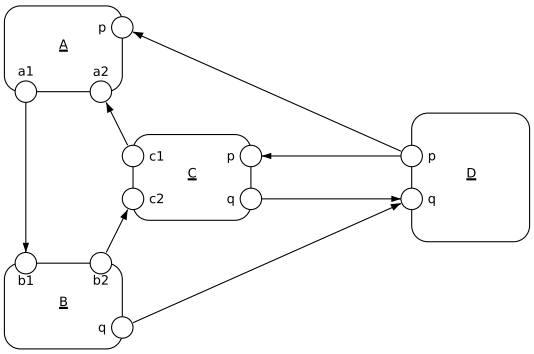


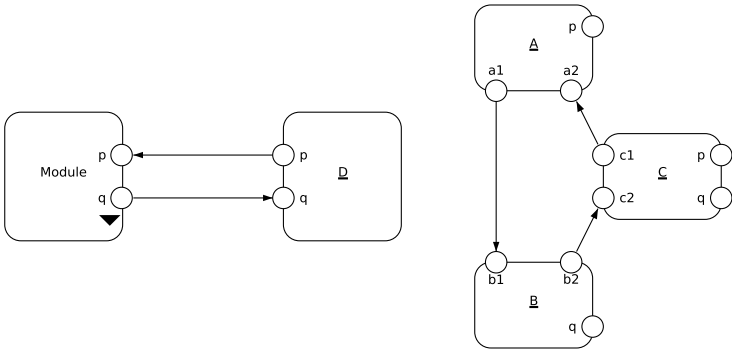
b)



c)

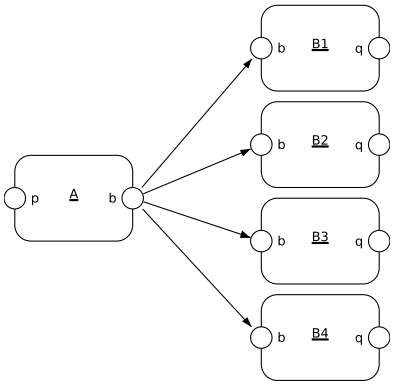
Użycie hierarchii – ograniczenie liczby połączeń

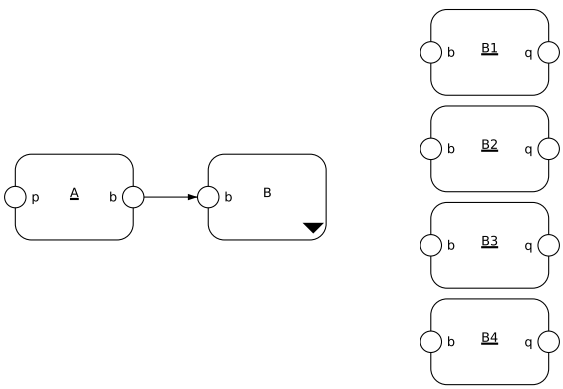




a) b)

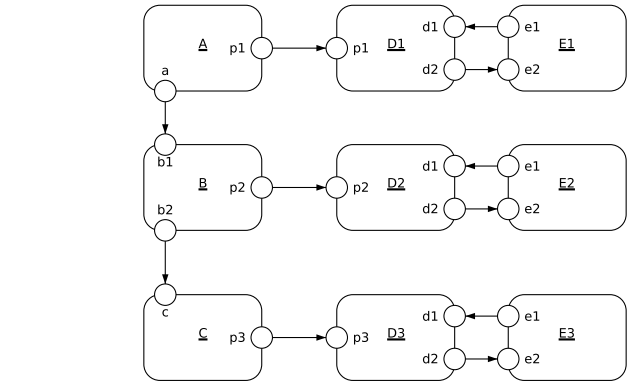
Użycie hierarchii – zarządzanie instancjami tego samego agenta

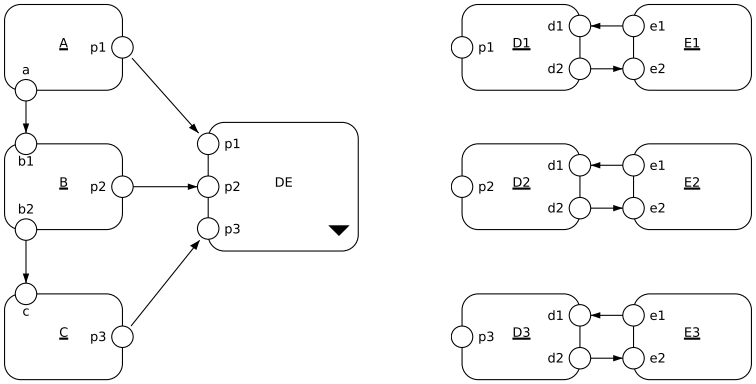




a) b)

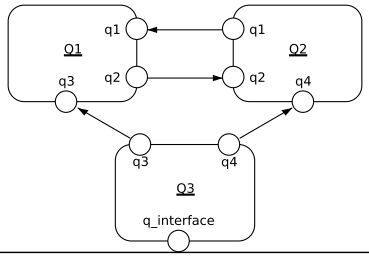
Użycie hierarchii – grupowanie powtarzających się fragmentów

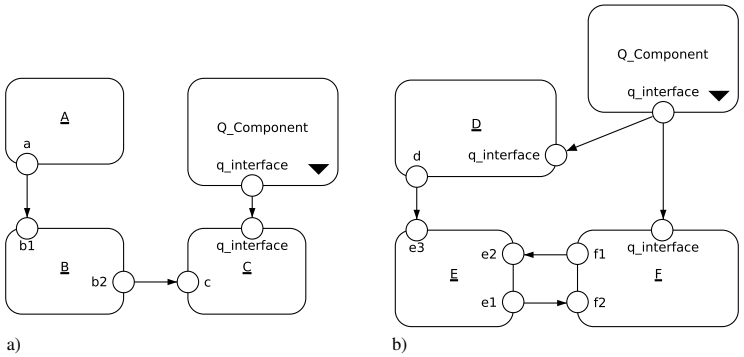




a) b)

Użycie hierarchii – wielokrotne użycie modułów





a) b)

Struktura warstwy kodu

```
-- Preamble:
--   types
--   constants
--   functions
--   environment specification

-- Implementation:
--   agents
```

```
agent AgentName(priority) {
-- declaration of parameters
-- agent body
}
```

Jeden blok definiujący dynamikę agenta może być współdzielony przez wiele agentów w modelu. W takim przypadku, po słowie kluczowym **agent** umieszczamy nazwy agentów oddzielone przecinkami. Wszystkie tak zdefiniowane agenty wykazują identyczne zachowanie.

Typy danych, parametry

- Alvis wykorzystuje system typów Haskella na potrzeby definiowania typów parametrów i modyfikowania ich wartości.
- Wybrane typy proste Haskella zalecane do stosowania w języku Alvis to: `Char`, `Bool`, `Int` i `Double`.
- Do najpopularniejszych typów złożonych Haskella należą listy i krotki.
  Lista jest ciągiem wartości tego samego typu, które są umieszczone w nawiasie kwadratowym i oddzielone przecinkami.
  Krotka może zawierać elementy różnych typów, a jej składowe są umieszczane w nawiasie okrągłym.

```
-- definicje stałych
size = 10;
name = "Agent";

--definicje parametrów
size      :: Int          = 7;
queue     :: [Double]     = [];
inputData :: (Int, Char) = (0, 'x');
```

Lista instrukcji – (1)

Instrukcja	Opis
`exec x = expression`	obliczenie wartości wyrażenia i przypisanie wyniku do parametru (można pominąć **exec**)
exit	zakończenie działania agenta aktywnego lub procedury agenta pasywnego
jump label	instrukcja skoku
loop {...}	pętla bezwarunkowa
loop (g) {...}	pętla warunkowa
null	instrukcja pusta
proc (g) p {...}	definicja procedury

select {
alt (g1) {...}
alt (g2) {...}
alt (g3) {...}
...
}
	instrukcja wyboru

start A	aktywowanie agenta aktywnego będącego w trybie *init*

Lista instrukcji (2)

Instrukcja	Opis
in p x	pobranie wartości przez port *p* i przypisanie jej do parametru *x* (wersja blokująca)
in (t) p x	pobranie wartości przez port *p* (wersja nieblokująca)
in (t) p x { **success** {...} } **fail** {...} }	pobranie wartości przez port *p* (wersja nieblokująca)
out p x	wysłanie wartości parametru (wartości) *x* przez port *p* (wersja blokująca)
out (t) p x	wysłanie wartości przez port *p* (wersja nieblokująca)
out (t) p x { **success** {...} } **fail** {...} }	wysłanie wartości przez port *p* (wersja nieblokująca)

W wersji rozbudowanej klauzule **success** i **fail** są opcjonalne. We wszystkich przedstawionych instrukcjach pominięcie *x* oznacza przesyłanie sygnału bez przypisanej konkretnej wartości.

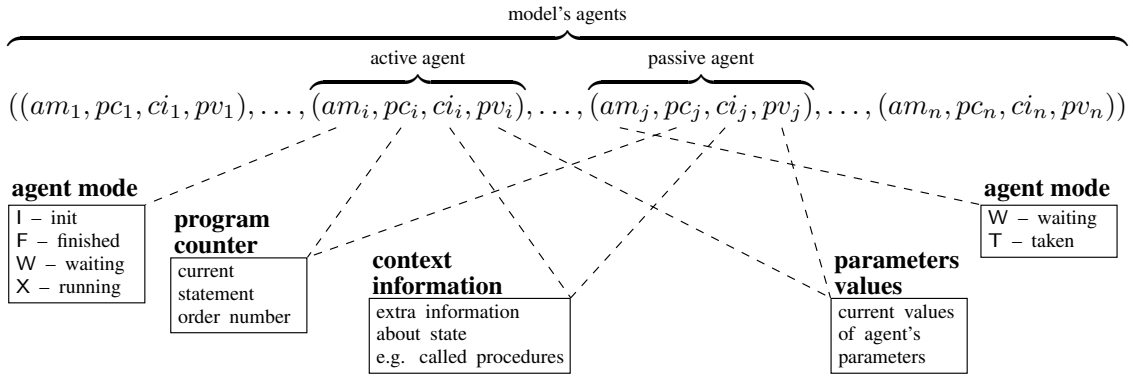
Warstwa kodu – komunikacja (1)

- Instrukcje **`in`** i **`out`** przyjmują jako obowiązkowy argument nazwę portu. Jeżeli przesyłane są dane, to drugim argumentem instrukcji **`in`** jest nazwa parametru, któremu przypisana zostaje pobrana wartość. W przypadku instrukcji **`out`** drugim argumentem może być nazwa parametru lub stała.
- Możliwość użycia portu jako argumentu instrukcji **`in`** i **`out`** zależy od tego, czy jest to port wejściowy, wyjściowy, czy też wejściowo-wyjściowy. Rola, jaką może pełnić port, zależy wyłącznie od połączeń tego portu w diagramie komunikacji.
- **Komunikacja między dwoma agentami aktywnymi może zostać zapoczątkowana przez dowolnego z nich.** Jeżeli komunikacja została zapoczątkowana wykonaniem instrukcji **`out`**, to agent, który ją wykonał, przechodzi w tryb oczekiwania, dopóki drugi z agentów nie pobierze wystawionej wartości. Jeżeli komunikacja została zapoczątkowana wykonaniem instrukcji **`in`**, to agent, który ją wykonał, przechodzi w tryb oczekiwania na dostarczenie odpowiedniej wartości.
- W przypadku instrukcji nieblokującej, oczekiwanym jest, że wykonanie instrukcji komunikacji kończy komunikację (lub jest wywołaniem dostępnej procedury). Jeżeli tak nie jest, to agent porzuca komunikację i wykonuje kolejne instrukcje.
- Do zrealizowania komunikacji nie wystarczy tylko połączenie w diagramie komunikacji. Dodatkowo wysyłana i odbierana wartość muszą być tego samego typu (w szczególności oba agenty mogą realizować komunikację bezargumentową).

Warstwa kodu – komunikacja (2)

- **Komunikacja między agentem aktywnym i pasywnym może być zapoczątkowana wyłącznie przez agenta aktywnego.**
- Procedury dzielimy na wejściowe i wyjściowe.
  W przypadku procedury wejściowej (dane wysyłane do agenta pasywnego), agent aktywny wywołuje taką procedurę, wykonując instrukcję **`out`**, co jednocześnie oznacza wysłanie argumentu do procedury. Jeżeli wywoływana procedura jest dostępna, to rozpoczyna się wykonywanie tej procedury i jest ono realizowane w kontekście agenta aktywnego. Agent pasywny odbiera przesyłany argument za pomocą instrukcji **`in`** (nie musi to być pierwsza instrukcja procedury). Argumentem tej instrukcji musi być wywołany port proceduralny. Wykonanie procedury kończy się wykonaniem instrukcji **`exit`**.
- W przypadku procedury wyjściowej (dane pobierane od agenta pasywnego), agent aktywny wywołuje taką procedurą wykonując instrukcję **`in`**. Agent pasywny zwraca wynik, wykonując instrukcję **`out`** (nie musi to być ostania instrukcja procedury). Argumentem instrukcji **`out`** musi być wywołany port proceduralny. Wykonanie procedury kończy się wykonaniem instrukcji **`exit`**.
- Agent pasywny może wywołać procedurę innego agenta pasywnego. Wywołanie takie musi być realizowane z użyciem portu nieproceduralnego. Wywołana procedura jest realizowana w tym samym kontekście co procedura, z której to wywołanie nastąpiło.

Stan modelu



- $am(X)$ – tryb agenta,
- $pc(X)$ – wartość licznika rozkazów,
- $ci(X)$ – lista kontekstowa,
- $pv(X)$ – krotka parametrów.

Stanem modelu $\overline{\mathbf{A}} = (D, B, \alpha^0)$, gdzie $D = (\mathcal{A}, \mathcal{C}, \sigma)$ i $\mathcal{A} = \{X_1, \ldots, X_n\}$, nazywamy krotkę:

$$S = (S(X_1), \ldots, S(X_n)). \tag{14}$$

Tryb agenta

finished F – tryb ten oznacza, że agent już zakończył swoją działalność.

init I – jest to domyślny tryb agentów aktywnych, które nie są aktywowane przy uruchomieniu modelu. Agent taki może być uaktywniony za pomocą instrukcji **`start`**.

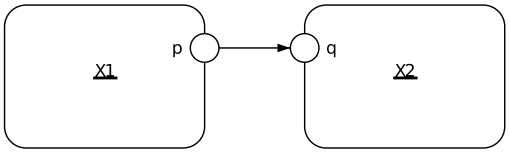
running X – tryb ten oznacza agenta, który wykonuje jedną ze swoich instrukcji. Agent aktywny pozostaje w trybie *running* również wtedy, gdy agent pasywny wykonuje procedurę w jego kontekście.

taken T – tryb ten oznacza, że agent pasywny jest w trakcie realizacji jednej ze swoich procedur.

waiting W – dla agentów pasywnych tryb ten oznacza, że dany agent jest bezczynny i czeka na wywołanie którejś z jego procedur. W takim trybie *lista kontekstowa* zawiera informacje o dostępnych procedurach (wraz z informacją, czy jest to procedura wejściowa, czy wyjściowa). Dla agentów aktywnych tryb ten oznacza, że dany agent albo czeka na sfinalizowanie komunikacji z innym agentem aktywnym, albo na aktualnie niedostępną procedurę agenta pasywnego.

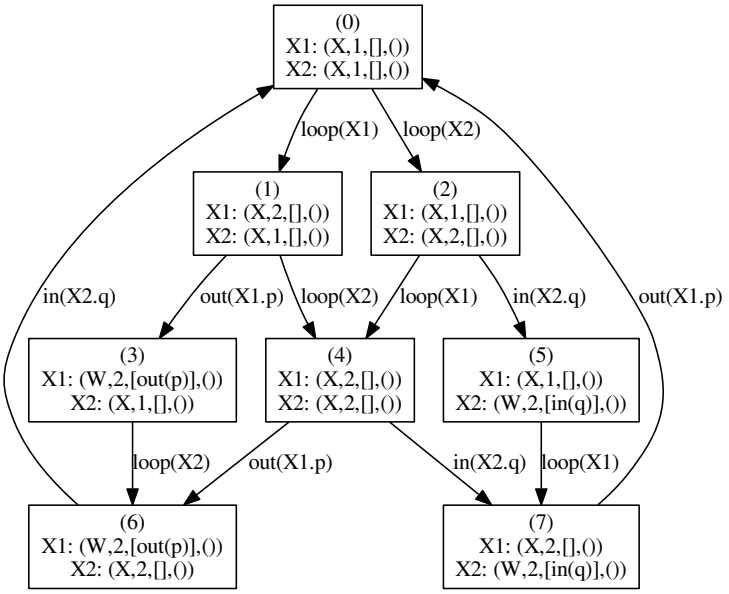
Jeżeli w kontekście agenta aktywnego wykonywana jest procedura agenta pasywnego, to tryb agenta aktywnego (możliwe są wartości X lub W) odnosi się do agenta pasywnego działającego w tym kontekście.

Alvis – graf LTS

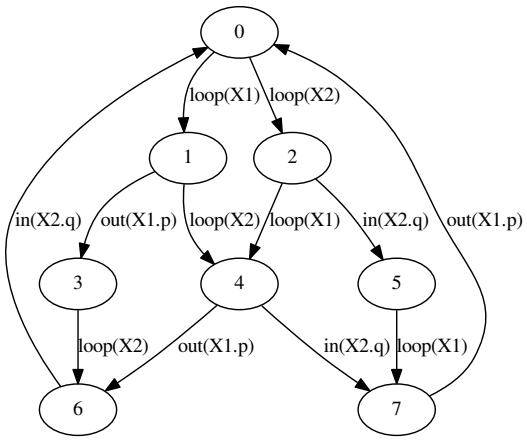



```
agent X1 {
  loop {
    out p; }
}
```

```
agent X2 {
  loop {
    in q; }
}
```



Graf LTS – weryfikacja z użyciem CADP




```
[true*] <true> true

nu X . <true> X

mu X . (<true> true and [not "in(X2.q)"] X)

[true* . "in(X2.q)"]
  mu X . (<true> true and [not "in(X2.q)"] X)

[true* . "in(X2.q)"] <(not "in(X2.q)")* .
  "in(X2.q)"> true

[true* . 'in(.*)' . (not 'out(.*)')* . 'in(.*)'] false
and
[true* . 'out(.*)' . (not 'in(.*)')* . 'out(.*)'] false

[true* . 'in(.*)' . (not 'out(.*)')* . 'in(.*)'
    . (not 'out(.*)')* . 'in(.*)'] false
and
[true* . 'out(.*)' . (not 'in(.*)')* . 'out(.*)'
    . (not 'in(.*)')* . 'out(.*)'] false
```