Agenty wnioskujące praktycznie (*Practical reasoning agents*)

Na podstawie "An Introduction to MultiAgent Systems" oraz slajdów Michaela Wooldridge'a

Czy jest wnioskowanie praktyczne (*practical reasoning*)?

- Wnioskowanie praktyczne dotyczy akcji – podjęcie decyzji, co zrobić na podstawie przekonań, zachcianek i intencji agenta (*beliefs* + *desires* + *intentions* = *BDI*)
- Różne od *wnioskowania teoretycznego*, dotyczącego przekonań, np. wszyscy ludzie są śmiertelni $\wedge$ Socrates jest człowiekiem $\rightarrow$ Socrates jest śmiertelny

Elementy wnioskowania praktycznego

Wnioskowanie praktyczne = rozważanie (*deliberation*) + planowanie (*means-ends reasoning*)

- Rozważanie – ustalenie, co chcemy osiągnąć → ***intencje*** (cele)
- Planowanie – ustalanie, jak osiągniemy nasze cele →**plany**

Obliczenia i ograniczenia zasobowe

Agent posiada ograniczone zasoby (pamięć, procesor)

- Zasoby narzucają ograniczenia na zakres obliczeń wykonywanych podczas wnioskowania
- Dodatkowe *ograniczenia czasowe* narzucone są przez rzeczywiste środowisko, w którym działa agent

Implikacje praktyczne

1. Agent musi kontrolować swój proces wnioskowania, jeśli ma „sensownie" działać
2. Agent nie może wnioskować w nieskończoność, nawet jeśli wybrane cele lub plany nie są optymalne

Intencje i zachcianki

Intencje w silniejszym stopniu wpływają na działanie/akcje, niż zachcianki...

Zachcianki to „dobre chęci”, natomiast intencje to chęci, które będą/są realizowane

Intencje we wnioskowaniu praktycznym

- Intencje sterują planowaniem
  - Jeśli agent ustalił intencję, wówczas próbuje ją zrealizować (zbudować plan)
  - Jeśli jedno podejście zawiedzie, wówczas agent próbuje innego
- Intencje trwają
  - Agent nie porzuca intencji bez dobrego powodu
  - Intencje trwają dopóki nie zostaną osiągnięte, agent stwierdzi, że nie może ich osiągnąć lub że ich powód przestał istnieć

Intencje we wnioskowaniu praktycznym

- Intencje ograniczają przyszłe kolejne rozważania
  - Agent nie powinien rozważać nowych zachcianek, które są sprzeczne z już przyjętymi
  - Intencje jako *filtr dopuszczalności*
- Intencje wypływają na przekonania uwzględnianie podczas wnioskowania
  - Jeśli agent przyjmuje intencję, wówczas powinien być przekonany, że ją zrealizuje
  - Z drugiej strony agent powinien dopuścić możliwość porażki

Reprezentacja przekonań, zachcianek i intencji

Agent musi jawnie reprezentować swoje przekonania, zachcianki i intencje (reprezentacja symboliczna np. w Prologu).
Niech

- B oznacza aktualne przekonania
- Bel oznacza zbiór wszystkich przekonać
- D oznacza aktualne zachcianki
- Des oznacza zbiór wszystkich zachcianek
- I oznacza aktualne intencje
- Int oznacza zbiór wszystkich intencji

Rozważanie

Etap rozważania modelowany za pomocą

1. funkcji generującej opcje

$$options : 2^{Bel} \times 2^{Int} \rightarrow 2^{Des}$$

2. funkcji filtrującej

$$filter : 2^{Bel} \times 2^{Des} \times 2^{Int} \rightarrow 2^{Int}$$

Aktualizacja przekonań agenta modelowana poprzez funkcję korekty przekonań (*belief revision function*)

$$brf : 2^{Bel} \times Per \rightarrow 2^{Bel}$$

Planowanie (means-ends reasoning)

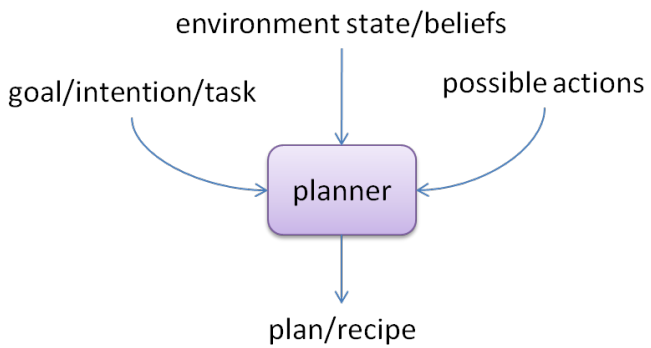
- Proces decydowania, jak osiągnąć **cel** (*end*) $\rightarrow$ intencję, którą przyjął agent, za pomocą dostępnych **środków** (*means*) $\rightarrow$ akcji, które agent może wykonać
- Planowanie jako synteza agenta lub (w ogólności) automatyczne programowanie

Planer

Planer to system, który na podstawie:

1. celu, intencji lub zadania (np. zadania osiągnięcia)
2. aktualnego stanu środowiska – przekonań agenta
3. akcji dostępnych dla agenta

generuje **plan** (sekwencję odpowiednich akcji)

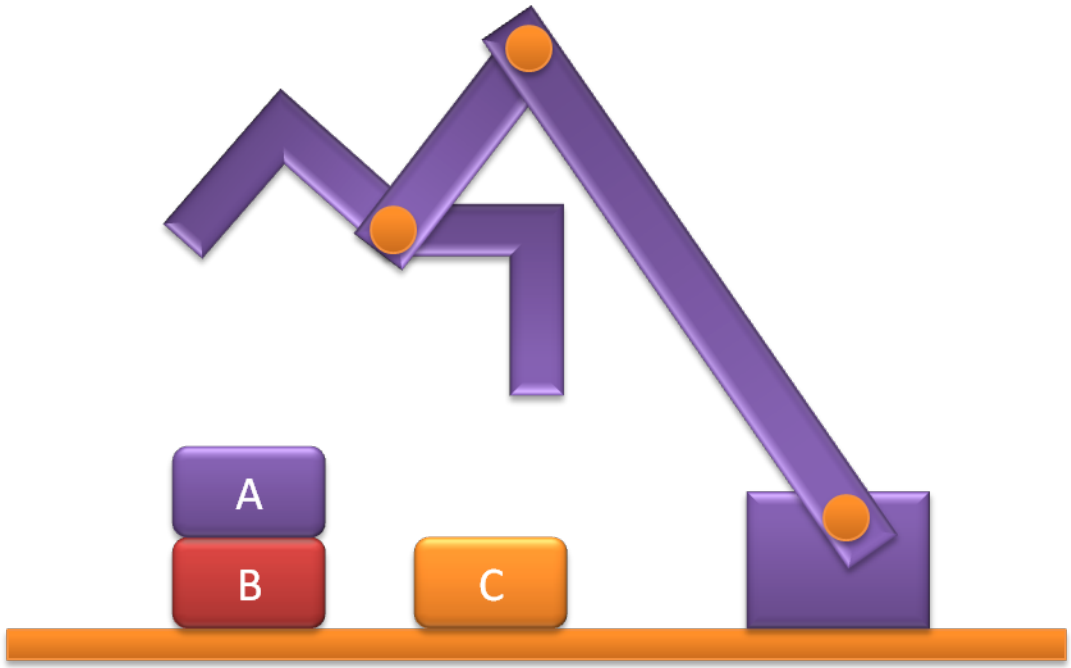


STRIPS

- STRIPS – pierwszy znany planer opracowany w latach 60/70–tych
- Dwa podstawowe elementy
  - model środowiska/świata (zbiór formuł logicznych)
  - zbiór schematów/definicji akcji opisujących ich warunki stosowania oraz efekty
- Idea algorytmu planowania
  - określenie różnicy między aktualnym stanem środowiska, a stanem docelowym
  - zredukowanie tej różnicy poprzez zastosowanie odpowiedniej akcji

Świat Klocków (the Blocks World)

Prosty robot (ramię z chwytakiem), klocki tego samego rozmiaru (A, B, C) , płaska powierzchnia (stół).



Reprezentacja w Świecie Klocków

Predykaty opisujące stan Świata Klocków

Predykat	Znaczenie
$On(x, y)$	klocek x na klocku y
$OnTable(x)$	klocek x na stole
$Clear(x)$	nie ma nic na klocku x
$Holding(x)$	ramię robota trzyma klocek x
$ArmEmpty$	ramię jest puste (nic nie trzyma)

Reprezentacja w Świecie Klocków

- Aktualny (początkowy) stan Świata Klocków (założenie świata zamkniętego)

$$\{Clear(A), On(A,B), OnTable(B), OnTable(C), Clear(C)\} \cup \\ \cup \{ArmEmpty\}$$

- Cel do osiągnięcia (intencja do zrealizowania)

$$\{OnTable(A), OnTable(B), OnTable(C)\}$$

Akcje w Świecie Klocków

Elementy opisujące akcję:

- *nazwa* – wraz z dodatkowym (opcjonalnymi) argumentami
- *lista warunków wstępnych* (*precondition list*) – lista faktów, które muszą być prawdziwe, aby akcja mogła być wykonana
- *lista do usunięcia* (*delete list*) – lista faktów, które przestają być prawdziwe po wykonaniu akcji
- *lista do dodania* (*add list*) – lista faktów, które stają się prawdziwe po wykonaniu akcji

Akcje w Świecie Klocków

- *Stack* – robot kładzie trzymany aktualnie klocek x na klocku y

$$Stack(x, y)$$

$$\begin{array}{ll} pre & \{Clear(y), Holding(x)\} \\ del & \{Clear(y), Holding(x)\} \\ add & \{ArmEmpty, On(x, y)\} \end{array}$$

- *UnStack* – robot podnosi klocek x z klocka y

$$UnStack(x, y)$$

$$\begin{array}{ll} pre & \{On(x, y), Clear(x), ArmEmpty\} \\ del & \{On(x, y), ArmEmpty\} \\ add & \{Holding(x), Clear(y)\} \end{array}$$

Akcje w Świecie Klocków

- *Pickup* – robot podnosi klocek x ze stołu

$$Pickup(x)$$

$$\begin{array}{ll} pre & \{Clear(x), OnTable(x), ArmEmpty\} \\ del & \{OnTable(x), ArmEmpty\} \\ add & \{Holding(x)\} \end{array}$$

- *PutDown* – robot kładzie aktualnie trzymany klocek x na stole

$$PutDown(x)$$

$$\begin{array}{ll} pre & \{Holding(x)\} \\ del & \{Holding(x)\} \\ add & \{ArmEmpty, OnTable(x)\} \end{array}$$

Planowanie bardziej formalnie

- Agent ma do dyspozycji ustalony (skończony) zbiór akcji $Ac = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$
- Deskryptor akcji $\alpha \in Ac$ to trójka $\langle P_\alpha, D_\alpha, A_\alpha \rangle$, gdzie
  - P_α jest *listą warunków wstępnych* dla α
  - D_α jest *listą do usunięcia* dla α
  - A_α jest *listą do dodania* dla α
- Problem planowania (nad zbiorem akcji Ac) jest opisany jako trójka $\langle \Delta, O, \gamma \rangle$, gdzie
  - Δ jest zbiorem przekonań agenta o początkowym stanie środowiska
  - $O = \{\langle P_\alpha, D_\alpha, A_\alpha \rangle \,|\, \alpha \in Ac\}$ jest zbiorem (indeksowanych) deskryptorów akcji
  - γ jest celem, zadaniem, intencją do zrealizowania lub osiągnięcia

Planowanie bardziej formalnie

- *Plan* π to sekwencja akcji $\pi = (\alpha_1, \alpha_2, \ldots, \alpha_m)$, gdzie $\alpha_i \in Ac$
- W kontekście problemu planowania $\langle \Delta, O, \gamma \rangle$ plan $\pi = (\alpha_1, \alpha_2, \ldots, \alpha_m)$ wyznacza sekwencję $m+1$ baz przekonań $\Delta_o, \Delta_1, \ldots, \Delta_m$, gdzie
  - $\Delta_o = \Delta$,
  - $\Delta_i = \left(\Delta_{i-1} \setminus D_{\alpha_i}\right) \cup A_{\alpha_i}$ dla $1 \leq i \leq m$

Planowanie bardziej formalnie

- Plan $\pi = (\alpha_1, \alpha_2, \ldots, \alpha_m)$ jest **akceptowalny** (*acceptable*) dla problemu $\langle \Delta, O, \gamma \rangle$ wtty, gdy warunki wstępne każdej akcji są spełnione przez poprzedzającą bazę przekonań, tzn. $\Delta_{i-1} \models P_{\alpha_i}$, $1 \leq i \leq m$
- Plan $\pi = (\alpha_1, \alpha_2, \ldots, \alpha_m)$ is **poprawny** (*correct*) dla problemu$\langle \Delta, O, \gamma \rangle$ wtty, gdy
  - π jest akceptowalny
  - $\Delta_n \models \gamma$

Zmodyfikowany problem dla planera

Dla danego problemu planowania $\langle \Delta, O, \gamma \rangle$, znajdź poprawny plan albo stwierdź, że on nie istnieje

Planowanie bardziej formalnie

Zdolność agenta do planowania reprezentowana przez funkcję

$$plan : 2^{Bel} \times 2^{Int} \times 2^{Ac} \rightarrow Plan$$

- Plan nie musi być budowany od podstaw – w wielu praktycznych zastosowaniach stosowane są biblioteki planów (lub podplanów)
- Znalezienie odpowiedniego planu oznacza znalezienie planu, którego warunku wstępne są zgodne z przekonaniami agenta, a warunki końcowe (*postconditions*) z intencją

Struktura sterująca agenta wnioskującego w sposób praktyczny

Struktura sterująca dla procesu podejmowania decyzji to pętla, w której agent

1. obserwuje świat i aktualizuje swoje przekonania
2. rozważa, jakie intencje powinien zrealizować (wcześniej generując zachcianki i filtrując je)
3. buduje plan pozwalający na realizację przyjętych intencji
4. wykonuje ten plan

Implementacja agenta wnioskującego w sposób praktyczny

Niech π będzie planem, a *Plans* będzie zbiorem wszystkich planów (nad zbiorem akcji *Ac*). Wtedy

- $pre(\pi)$ oznacza warunku wstępne planu π, a $body(\pi)$ oznacza „ciało" planu π
- $empty(\pi)$ oznacza sprawdzenie, czy plan π jest pusty, czy nie
- $execute(\pi)$ oznacza wykonanie planu π bez zatrzymywania się
- $hd(\pi)$ oznacza plan składający się z pierwszej akcji z planu π
- $tail(\pi)$ oznacza plan składający się ze wszystkich akcji plany π poza pierwszą
- $correct(\pi, I, B)$ oznacza sprawdzenie, czy plan π jest poprawny biorąc pod uwagę intencje I oraz przekonania B

Implementacja agenta wnioskującego w sposób praktyczny

```
1  B ← B0
2  I ← I0
3  while true do
4  |    get next percept ρ through see(...) function
5  |    B ← brf(B, ρ)
6  |    D ← options(B, I)
7  |    I ← filter(B, D, I)
8  |    π ← plan(B, I, Ac)
9  |    while not (empty(π) or succeeded(I, B) or impossible(I, B)) do
10 |    |    π ← hd(π)
11 |    |    execute(π)
12 |    |    π ← tail(π)
13 |    |    get next percept ρ through see(...) function
14 |    |    B ← brf(B, ρ)
15 |    |    if reconsider(I, B) then
16 |    |    |    D ← options(B, I)
17 |    |    |    I ← filter(B, D, I)
18 |    |    end
19 |    |    if not correct(π, I, B) then
20 |    |    |    π ← plan(B, I, Ac)
21 |    |    end
22 |    end
23 end
```

Zaangażowanie w intencje i plany

- Po wybraniu intencji przez agenta, agent zobowiązuje się ją osiągnąć (lub angażuje w jej osiągnięcie, *committment*)
- Zobowiązanie oznacza *czasową* trwałość – przyjęta intencja nie powinna być zbyt szybko porzucona

Jak bardzo agent powinien się zaangażować?

- jak trwała powinna być intencja?
- w jakich warunkach agent powinien ją porzucić?

Strategie zaangażowania

Strategia zaangażowana określa, jak długo agent powinien utrzymywać intencję

- Ślepe zaangażowanie (*blind*)
  - agent utrzymuje intencję aż dojdzie do przekonania, że została ona zrealizowana (zaangażowanie fanatyczne)
- Zaangażowanie jednostkowe (*single-minded*)
  - agent utrzymuje intencję aż dojdzie do przekonania, że została ona osiągnięta albo że nie można jej osiągnąć
- Zaangażowanie otwarte (*open-minded*)
  - agent utrzymuje intencję tak długo, jak panuje przekonanie, że można ją osiągnąć

Ponowne rozważenie intencji

Kiedy agent powinien wstrzymać wykonywanie planu i ponownie rozważyć intencje?

- Agent, który nie kontroluje intencji dostatecznie często, może wykonywać niepotrzebnie plan, np. jeśli niemożliwe stanie się zrealizowanie intencji
- Ale agent, który cały czas rozważa intencję, może nie mieć czasu na realizację planu i w efekcie nie zrealizuje intencji

Optymalne ponowne rozważenie intencji

- Eksperymentalna ocena różnych strategii ponownego rozważania intencji
- Dwie klasy agentów
  - agenci *odważni*, którzy nigdy nie rozważają ponownie intencji
  - agencji ostrożni, którzy rozważają intencję po każdej akcji
- Dynamizm środowiska reprezentowany przez wskaźnik γ – liczba zmian środowiska przypadająca na jedna iterację pętli działania agenta
- Efektywność agenta wyznaczona jako stosunek zrealizowanych intencji do wszystkich przyjętych

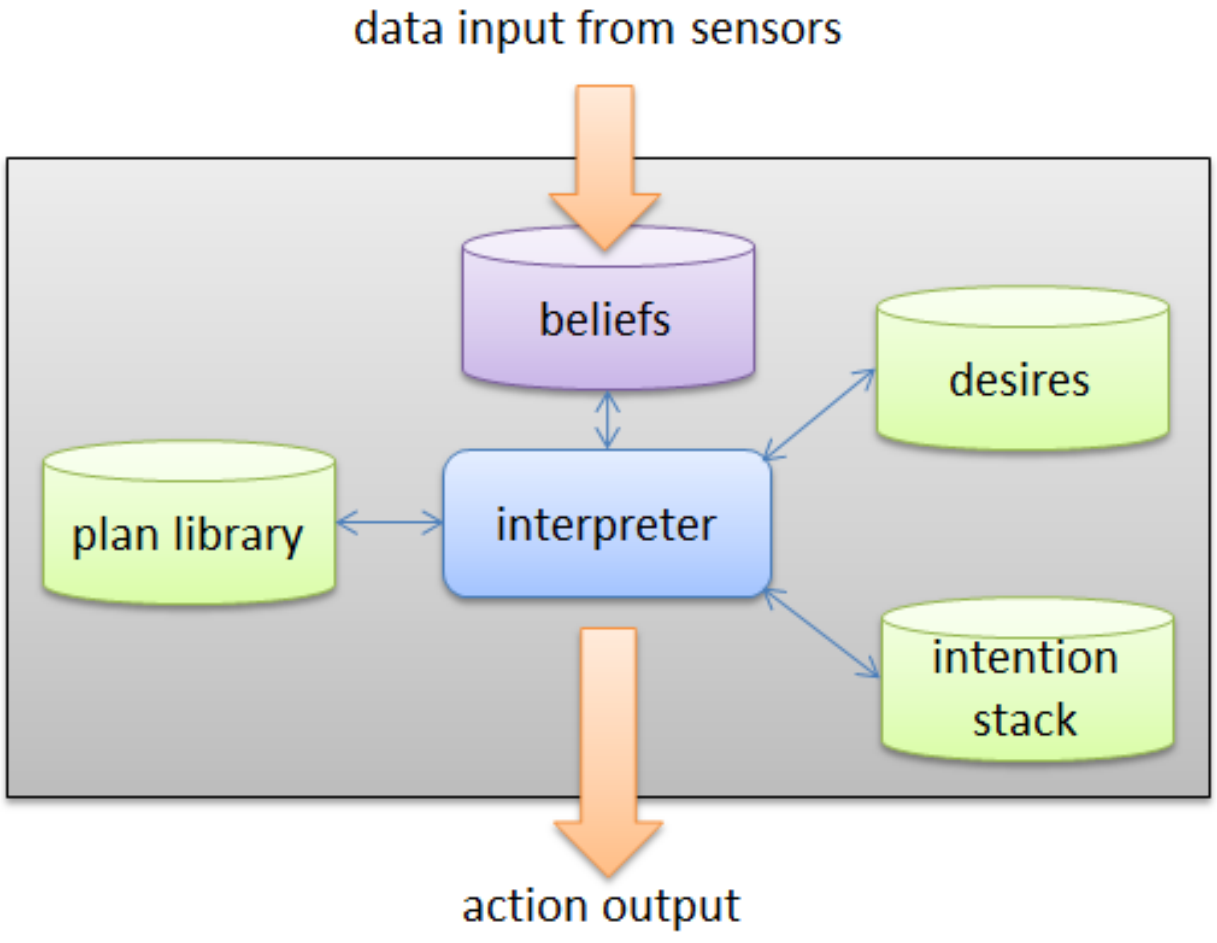
Optymalne ponowne rozważenie intencji

- Jeśli γ jest małe (środowisko stabilne), wówczas agenty odważne są bardziej efektywne, niż ostrożne – nie tracą czasu na ponowne zastanawianie się
- Jeśli γ jest duże (środowisko dynamiczne), wówczas agenty ostrożne są bardziej efektywne – szybciej wykrywają konieczność zmiany planu

The Procedural Reasoning System (PRS)

- PRS to jedna z pierwszych (i najbardziej trwałych) architektur agentowych wykorzystujących wnioskowanie praktyczne oraz BDI (lata 80-te)
- Wykorzystana w kilku znanych i bardzo złożonych systemach wieloagentowych (kamienie milowe AI)
  - OASIS – system do zarządzania ruchem powietrznym (lotnisko w Sydney)
  - SWARMM – system symulacyjno-treningowy dla RAAF (Royal Australian Air Force)

Architektura PRS



Plany w PRS

- Biblioteka gotowych (pod)planów
- Każdy plan składa się z następujących komponentów
  - *cel* (*goal*) – warunek zakończenia
  - *kontekst* (*context*) – warunek wstępny (warunek zastosowania)
  - *ciało* (*body*)– sekwencja akcji do wykonania
- Plany mogą definiować dodatkowe cele – cele te muszą być osiągnięte przed wykonaniem reszty plan ($\rightarrow$ wywołanie podplanów)
- Możliwość zaawansowanego definiowania celów – dysjunkcje ('achieve ϕ or achieve ψ') i pętle ('keep achieving ϕ until ψ')

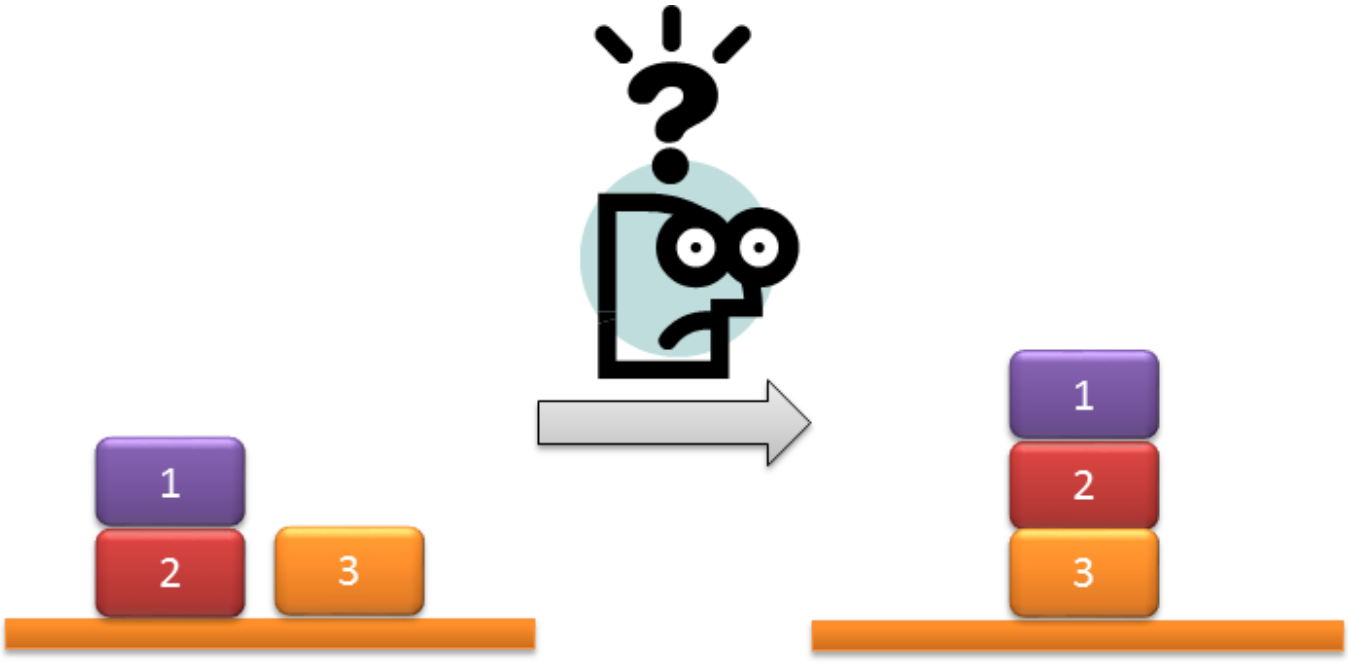
Planowanie w PRS

- Gdy agent startuje, umieszcza swoją intencję (cel) na stosie celów (*intention stack*)
- Agent przeszukuje bibliotekę planów w poszukiwaniu tych, których cel jest taki sam, jak cel agenta
- Agent sprawdza warunki wstępne znalezionych planów – plany ze spełnionymi warunkami traktowane są jako opcje (zachcianki)

Planowanie w PRS

- Podczas fazy rozważania agent wybiera najlepszy plan spośród dostępnych opcji
  - meta-plany (plany dotyczące planów)
  - maksymalizacja użyteczności związanych z poszczególnymi planami
- Wybrany plan jest wykonywany – może to powodować wstawianie dodatkowych celów na stos intencji
- Jeśli dany plan zakończy się niepowodzeniem, agent może wybrać kolejną ze zidentyfikowanych opcji

Świata Klocków znów...



Realizacja w JAM – implementacji PRS w języku Java

Przykład w JAM

```
GOALS:
    ACHIEVE blocks_stacked;
FACTS:
    // Block1 on Block2 initially so need to clear Block2 before stacking.
    FACT ON "Block1" "Block2";
    FACT ON "Block2" "Table";
    FACT ON "Block3" "Table";
    FACT CLEAR "Block1";
    FACT CLEAR "Block3";
    FACT CLEAR "Table";
```

Przykład w JAM

```
Plan: {
NAME: "Top-level plan"
DOCUMENTATION: "Establish Block1 on Block2 on Block3."
GOAL:
    ACHIEVE blocks_stacked;
CONTEXT:
BODY:
    EXECUTE print "Goal is Block1 on Block2 on Block2 on Table.\n";
    EXECUTE print "World Model at start is:\n"; EXECUTE printWorldModel;
    EXECUTE print "ACHIEVEing Block3 on Table.\n";
    ACHIEVE ON "Block3" "Table";
    EXECUTE print "ACHIEVEing Block2 on Block3.\n";
    ACHIEVE ON "Block2" "Block3";
    EXECUTE print "ACHIEVEing Block1 on Block2.\n";
    ACHIEVE ON "Block1" "Block2";
    EXECUTE print "World Model at end is:\n";
    EXECUTE printWorldModel;
}
```

Przykład w JAM

```
Plan: {
NAME: "Stack blocks that are already clear"
GOAL:
     ACHIEVE ON $OBJ1 $OBJ2;
CONTEXT:
BODY:
     EXECUTE print "Making sure " $OBJ1 " is clear\n";
     ACHIEVE CLEAR $OBJ1;
     EXECUTE print "Making sure " $OBJ2 " is clear.\n";
     ACHIEVE CLEAR $OBJ2;
     EXECUTE print "Moving " $OBJ1 " on top of " $OBJ2 ".\n";
     PERFORM move $OBJ1 $OBJ2;
UTILITY: 10;
FAILURE:
     EXECUTE print "\n\nStack blocks failed!\n\n"; }
}
```

Przykład w JAM

```
Plan: {
NAME: "Clear a block"
GOAL:
     ACHIEVE CLEAR $OBJ;
CONTEXT:
     FACT ON $OBJ1 $OBJ;
BODY:
     EXECUTE print "Clearing " $OBJ1 " from on top of " $OBJ "\n";
     EXECUTE print "Making sure " $OBJ1 " is clear\n";
     ACHIEVE CLEAR $OBJ1;
     EXECUTE print "Moving " $OBJ1 " to table.\n";
     PERFORM move $OBJ1 "Table";
EFFECTS:
     EXECUTE print "CLEAR: Retracting ON " $OBJ1 " " $OBJ "\n";
     RETRACT ON $OBJ1 $OBJ;
FAILURE:
     EXECUTE print "\n\nClearing block " $OBJ " failed!\n\n";
}
```

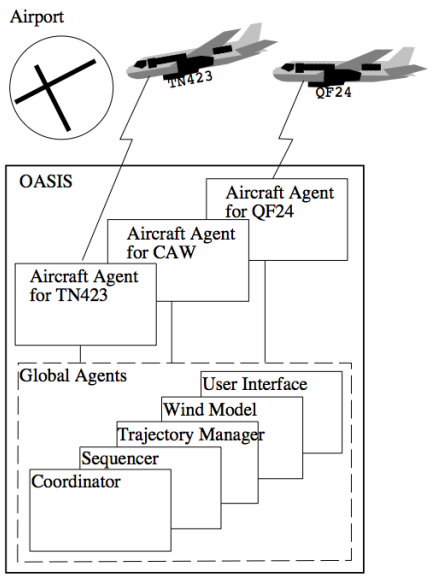
Przykład w JAM

```
Plan: {
NAME: "Move a block onto another object"
GOAL:
    PERFORM move $OBJ1 $OBJ2;
CONTEXT:
    FACT CLEAR $OBJ1;
    FACT CLEAR $OBJ2;
BODY:
    EXECUTE print "Performing low-level move action";
    EXECUTE print " of " $OBJ1 " to " $OBJ2 ".\n";
EFFECTS:
    WHEN : TEST (!= $OBJ2 "Table") {
        EXECUTE print " Retracting CLEAR " $OBJ2 "\n";
        RETRACT CLEAR $OBJ2;
    }
    FACT ON $OBJ1 $OBJ3;
    EXECUTE print " move: Retracting ON " $OBJ1 " " $OBJ3 "\n";
    RETRACT ON $OBJ1 $OBJ3;
    EXECUTE print " move: Asserting CLEAR " $OBJ3 "\n";
    ASSERT CLEAR $OBJ3;
    EXECUTE print " move: Asserting ON " $OBJ1 " " $OBJ2 "\n\n";
    ASSERT ON $OBJ1 $OBJ2;
FAILURE:
    EXECUTE print "\n\nMove failed!\n\n";
};
```

System OASIS

- System wspomagający planowanie odpowiedniej sekwencji lądowania
  - Maksymalizacja wykorzystania pasów
  - Ustalanie optymalnej sekwencji lądowania samolotów, przypisanie czasów lądowania i monitorowanie samolotów
- Plany zwyczajowo przygotowywane przez *Flow Directors*
  - Planowanie rozpoczyna się ~ 120 mil / 20 minut przed lądowaniem
  - Dotyczy jedynie sekwencji lądowania – kwestie związane z bezpieczeństwem obsługiwane przez kontrolerów lotu
- System przygotowany dla lotniska w Sidney (do 60 lądowań na godzinę)

Struktura systemu OASIS



- *Coordinator* – koordynuje pracę innych agentów globalnych
- *Sequencer* – definiuje sekwencję lądowania
- *Trajectory Manager* – weryfikuje, czy sekwencja opracowana przez *Sequencer*-a nie narusza wymagań bezpieczeństwa
- *Wind Model* – aktualizuje model wiatru na podstawie obserwacji z poszczególnych samolotów
- *User Interface* – pośredniczy między *Flow Director*-em a pozostałymi elementami systemu
- *Aircraft* – określa czas lądowania, monitoruje przebieg lotu oraz planuje ostatni etap lotu

Agent *Sequencer*

- Identyfikacja „zatorów” – przerwy między lądowaniami krótsze niż dopuszczalne
- Ustalanie alternatywnych czasów lądowania dla wybranych samolotów – przeszukiwanie z ograniczeniami
  - dostępność i wydolność pasów do lądowania (zależna od pogody)
  - minimalne przerwy między samolotami (zależne np. od wielkości samolotów – dłuższa przerwa po dużym samolocie)
  - poziomy paliwa poszczególnych samolotów
  - osiągi poszczególnych samolotów (np. zdolność do zwiększania prędkości)
- Proces przeszukiwania może zostać przerwany w dowolnym momencie – zwracane jest wtedy najtańsze (całkowite opóźnienie ważone priorytetami poszczególnych lotów) dotychczas znalezione rozwiązanie

Agent *Aircraft*

- Moduł *Predictor*
  - oblicza czas lądowania na lotnisku (min...max)
  - uwzględnia możliwości samolotu oraz informację o wietrze od agenta *Wind Model*
- Moduł *Monitor*
  - porównuje przewidywania *Predictor*-a z aktualnym czasem lotu
  - informuje moduł Planner oraz agenta Sequencer o rozbieżnościach
  - wysyła dane do agenta *Wind Model*, aby zaktualizować model wiatru
- Moduł *Planner*
  - przygotowuje plan dla samolotu, aby zapewnić lądowanie o wyznaczonym czasie
  - korzysta z biblioteki gotowych planów (dokładnych oraz heurystyk)
  - opcje prezentowane są *Flow Director*-owi, który wybiera ostateczny plan do realizacji

OASIS - przykład działania

W stronę lotniska SY lecą 3 samoloty – S1, S2 i N. Wszystkie będą przelatywały nad radiolatarnią BIK.

Wstępnie raportowane czasy lądowania (wyznaczone przez moduły *Predictor* agentów obsługujących te samoloty)

Samolot	ETA @ BIK	ELT @SY	Min	Max
S1	00:50	01:01	01:00	01:30
N	00:49	01:02	01:01	01:40
S2	00:52	01:03	1:02	01:40

OASIS - przykład działania

Sequencer ustala następującą sekwencję zwiększającą odstępy między lądowaniami (opóźnienie - 4 min) i przesyła ją samolotom

- S1: *Planner* zwiększa prędkość o 20 węzłów
- N: *Planner* proponuje zmniejszyć prędkość lub wydłużyć trasę – *Flow Director* wybiera drugą opcję
- S2: *Planner* proponuje zmniejszyć prędkość lub wydłużyć trasę – *Flow Director* wybiera drugą opcję

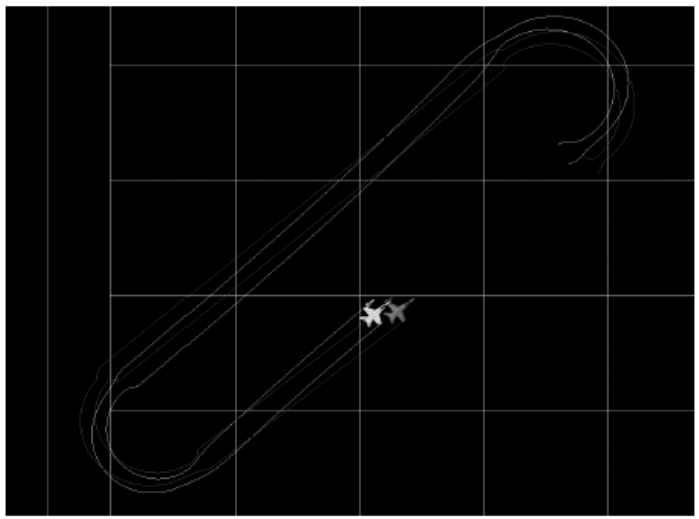
Samolot	ALT	Zmiana ELT	Prędkość	Zmiana trasy	ETA @ BIK
S1	1:00	-1	320	0	00:49
N	1:03	+1	280	+7.5	00:50
S2	1:06	+3	260	+23.0	00:55

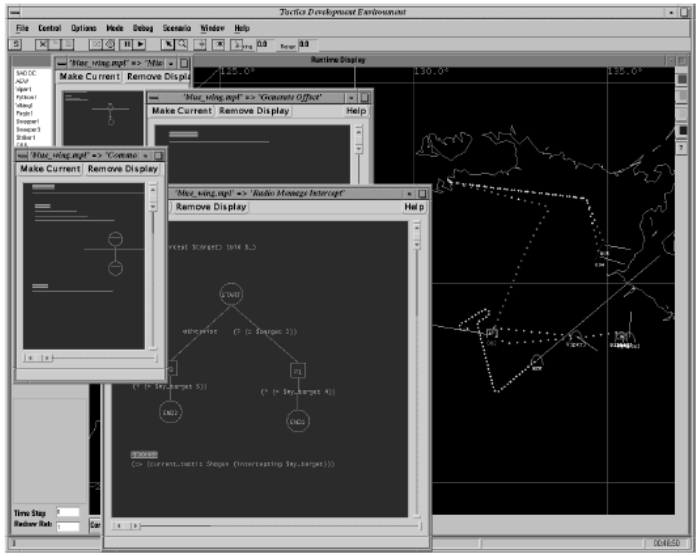
N przelatuje nad BIK o 00:51 – *Monitor* wykrywa różnicę i raportuje *Planner*-owi, który zwiększa prędkość do 300 węzłów. W przypadku większych odchyleń konieczna byłaby zmiana sekwencji przez *Sequencer*-a.

System SWARMM

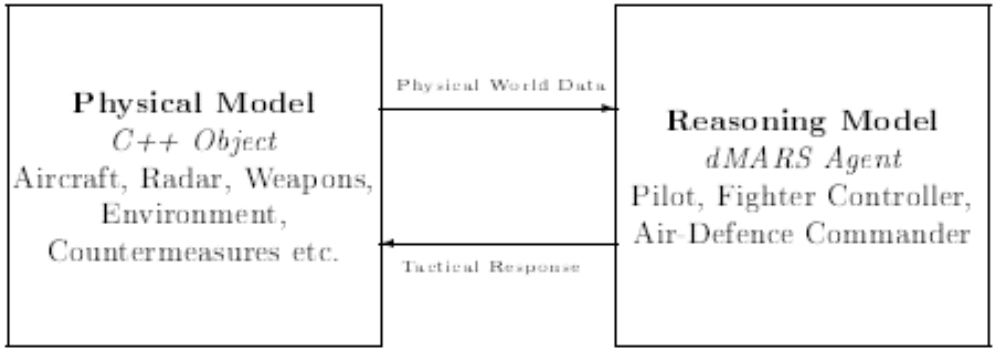
- System służący do modelowania i symulowania procesów podejmowania decyzji przez pilotów myśliwców i kontrolerów biorących udział w walkach powietrznych
- Wykorzystany także do oceny nowych taktyk oraz do treningu pilotów (HIL, human-in-the-loop) – symulowani przeciwnicy
- Zdolność do wyjaśniania podejmowanych decyzji i prezentowanego zachowania
  - wykorzystanie wiedzy eksperckiej
  - ale problemy z jej pozyskaniem ($\rightarrow$ *experts just know*)
- Stworzony i wykorzystywany przez Royal Australian Air Forces

SWARMM – przykłady UI





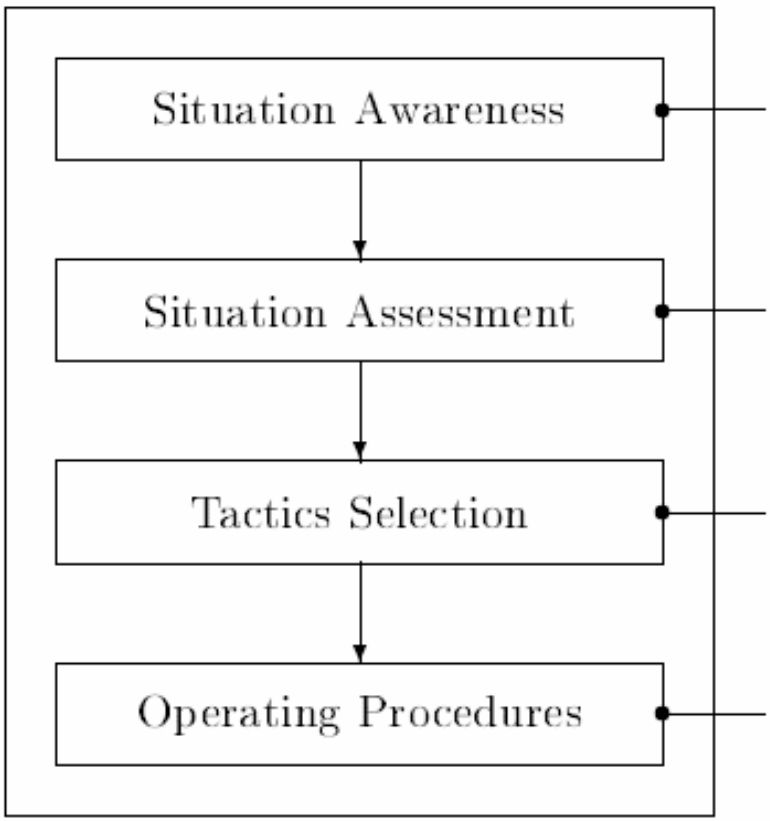
Struktura SWARMM



- *Modele fizyczne* (agenty zewnętrzne) symulujący zachowanie obiektów z punktu widzenia „fizyki”
- *Modele wnioskujące* podejmujące decyzje
- Modele fizyczne dostarczają informacji modelom wnioskującym (podział kompetencji)

Projekt agenta

Proces podejmowania decyzji przez agenta-pilota



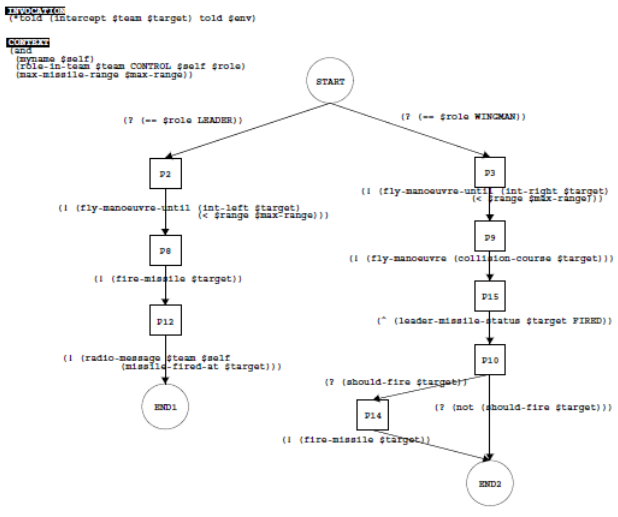
Takes data from the physical models. Decides what should be reacted to. Forms more complex symbolic descriptors from the raw data

Recognizes the situation from the description that comes out of the previous step. Based on the raw data and the output from the SA module this module assesses and characterizes the situaiton

Based on the recognized situations from the previous step this module selects a tactic based on the current set of goals and intentions and anything new that has emerged

Implements the tactical decision as one of the available standard operating procedures Choice is based on a number of factors including goals, team structures, threats, resources etc.

Plany



- 500 planów dla agenta-pilota
- Plany przygotowane we współpracy z pilotami myśliwców
- Łatwa modyfikacja/aktualizacja w trakcie działania systemu