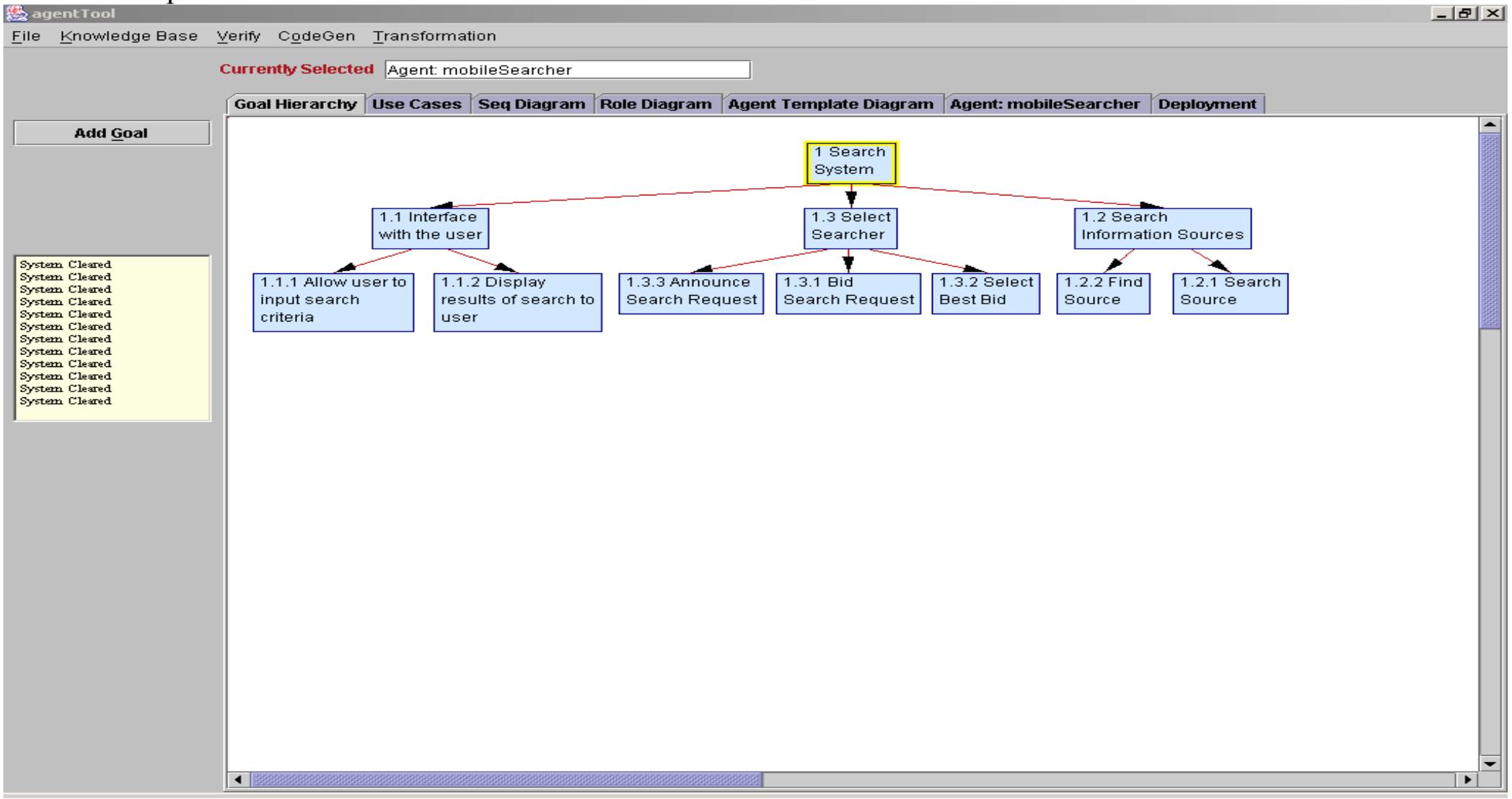
Wykład 3_1
Przykład aplikacji: Wyszukiwanie informacji w internecie

I. Analiza problemu



Rys. 8.1. Diagram typu *Goal*

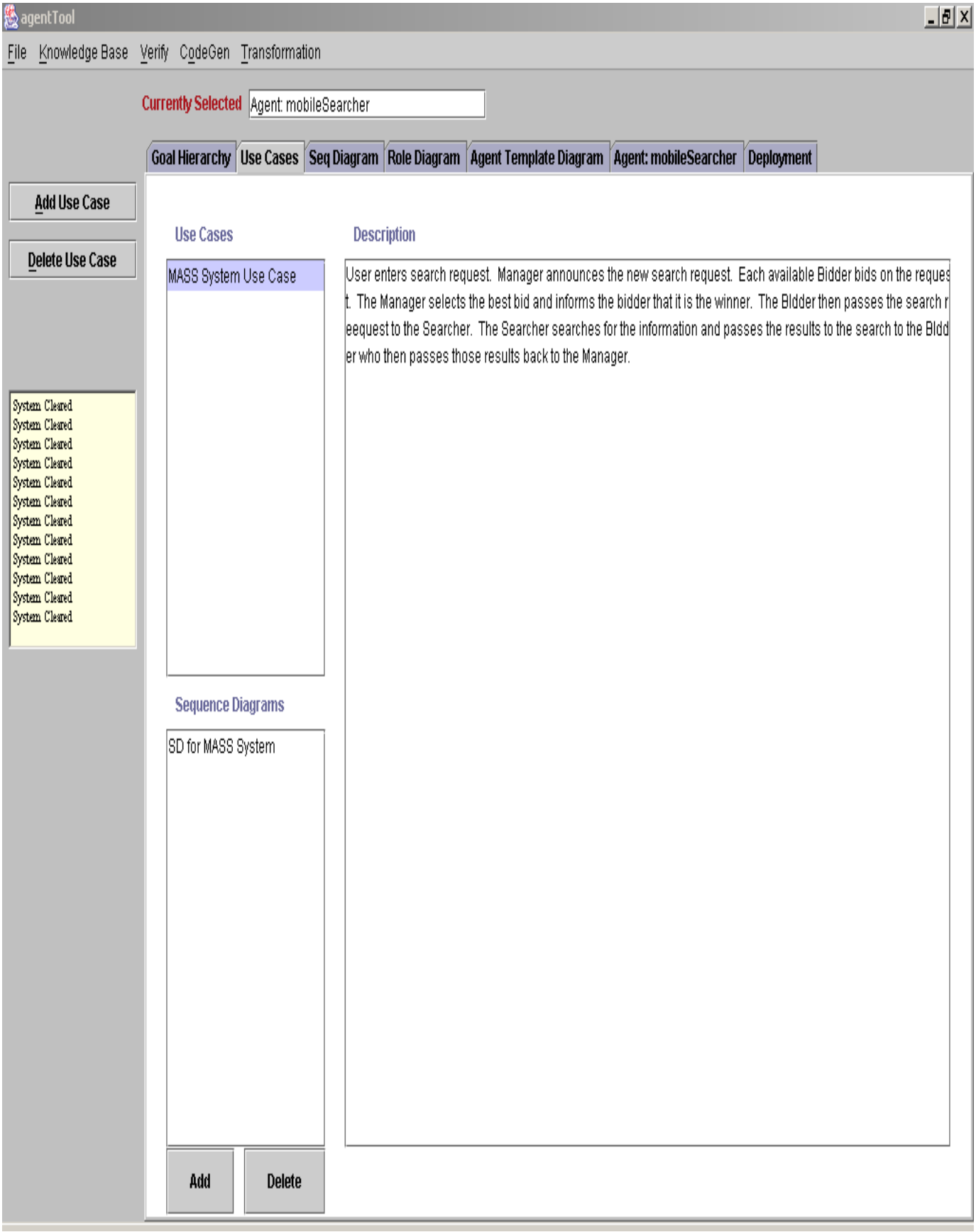
1. Opis diagramu typu *Goal*

- Główne zadanie systemu polega na efektywnym wyszukaniu informacji w plikach o określonych formatach np. txt, doc, html itd.
- Aplikacja zawiera dwa poziomy podzadań

1. System wyszukiwania
1.1. Kontakt z użytkownikiem
1.1.1. Wprowadzanie kryteriów wyszukiwania przez użytkownika
1.1.2. Wyświetlenie wyników wyszukiwania użytkownikowi
1.2. Przeszukiwanie źródeł informacji
1.2.1. Poszukiwanie informacji
1.2.2. Przeszukiwanie informacji
1.3. System wyboru wyszukiwarki
1.3.1. Zgłoszenie o zapotrzebowania na wyszukanie informacji
1.3.2. Licytacja ofert wyszukiwania
1.3.3. Wybór najlepszej wyszukiwarki spełniającej kryteria

2. Diagram Use-Case

2.1.Użytkownik wprowadza polecenie.
2.2. Rola *Manager* rozpoczyna akcję poszukiwania informacji tzn. zgłasza potrzebę wyszukiwania informacji i ogłasza licytację propozycji wyszukiwania
2.3. Każda dostępna rola *Bidder* podaje roli *Manager* swoje propozycje poszukiwania.
2.4. *Manager* wybiera najlepszą rolę *Bidder*.
2.5. Zwycięzca przekazuje polecenie poszukiwania pełniącemu rolę *Searcher*
2.6. *Searcher* wyszukuje informację i przekazuje ją roli *Bidder*, która przekazuje tę informację roli *Manager*.



Rys. 8.2. Diagram typu Use-Case

3. Opis diagramu interakcji

3.1. Rola *mgr:Manager* wysyła zgłoszenie zapotrzebowania na wyszukiwanie informacji (*Annouce search request*) do ról *bidder(1):Bidder* i *bidder(2):Bidder*

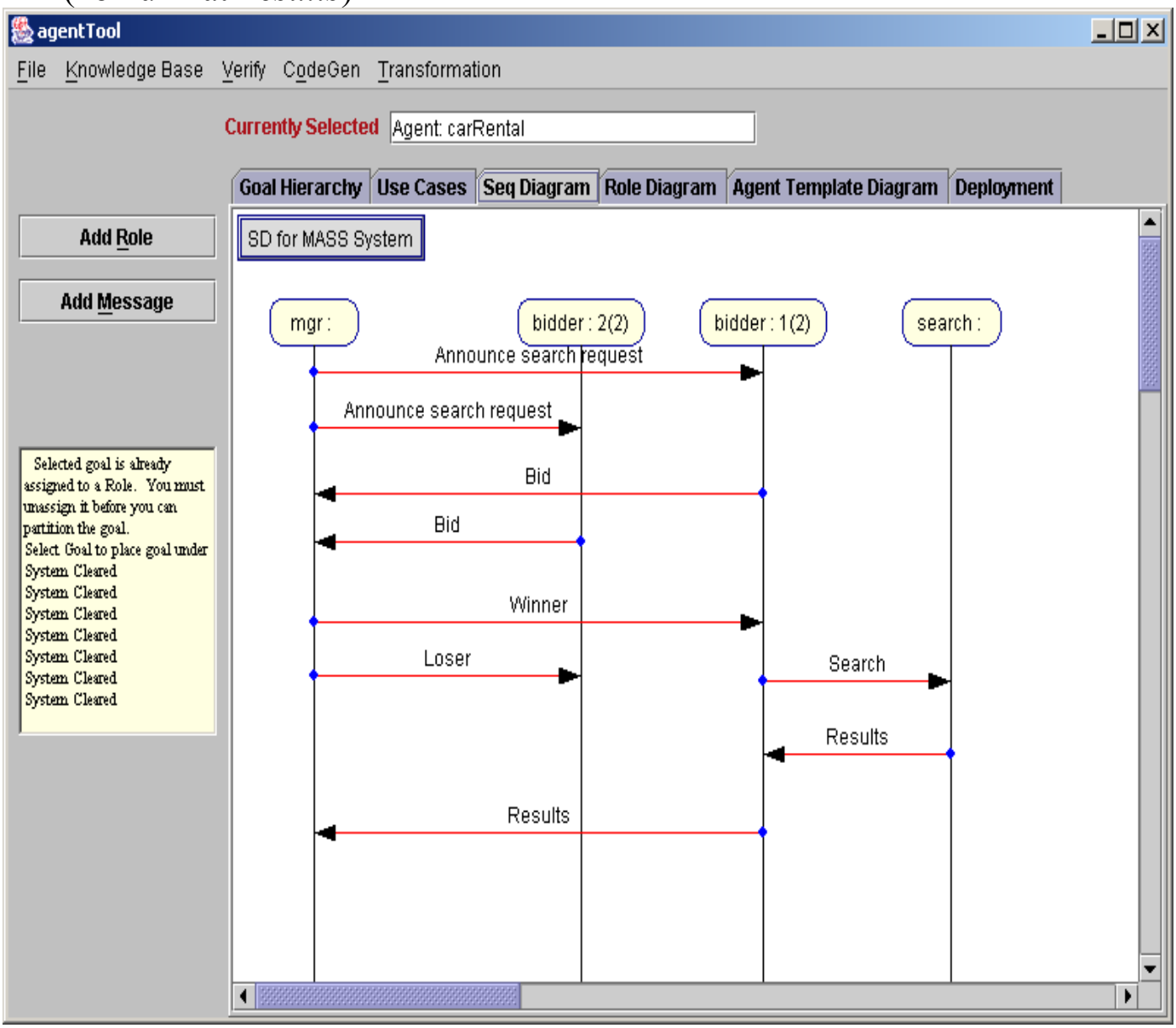
3.2. Role *bidder(1):Bidder* i *bidder(2):Bidder* podają oferty (*Bid*) wyszukiwania roli *mgr:Manager*

3.3. Rola *mgr:Manager* wybiera najlepszą ofertę wg określonego algorytmu (np. Sztuczna Inteligencja) i wysyła komunikat *Winner* wybranej roli *bidder:Bidder* oraz komunikaty *Loser* odrzuconym rolom *bidder:Bidder*

3.4. Wybrana rola *bidder:Bidder* wysyła rolom *search:Search* polecenie wyszukiwania *search:Search*

3.5. Role *search:Search* po wykonaniu wyszukiwania podają rezultat wyszukiwania wybranej roli *bidder:Bidder* (komunikat *Results*)

3.6. Rola *bidder:Bidder* podaje rezultat wyszukiwania roli *mgr:Manager* (komunikat *Results*)



Rys.8.3. Diagram sekwencji

4. Diagram ról

Diagram ról powstaje na podstawie informacji pozyskanych z diagramów typu *Goal* oraz z diagramu *Use-Case* i diagramów sekwencji

4.1. Wyodrębnione trzy role na diagramie Use-Case realizują następujące cele:

- Rola *mgr:Manager* realizuje cele:
  1.1.Kontakt z użytkownikiem
  1.1.1. Wprowadzanie kryteriów wyszukiwania przez użytkownika
  1.1.2. Wyświetlenie wyników wyszukiwania użytkownikowi
- Rola *bidder:Bidder* realizuje cele:
  1.3. System wyboru wyszukiwarki
  1.3.1.Zgłoszenie o zapotrzebowania na wyszukanie informacji
  1.3.2.Licytacja ofert wyszukiwania
  1.3.3.Wybór najlepszej wyszukiwarki spełniającej kryteria
- Rola *search:Search* realizuje cele:
  1.2. Przeszukiwanie źródeł informacji
  1.2.1. Poszukiwanie informacji
  1.2.2. Przeszukiwanie informacji

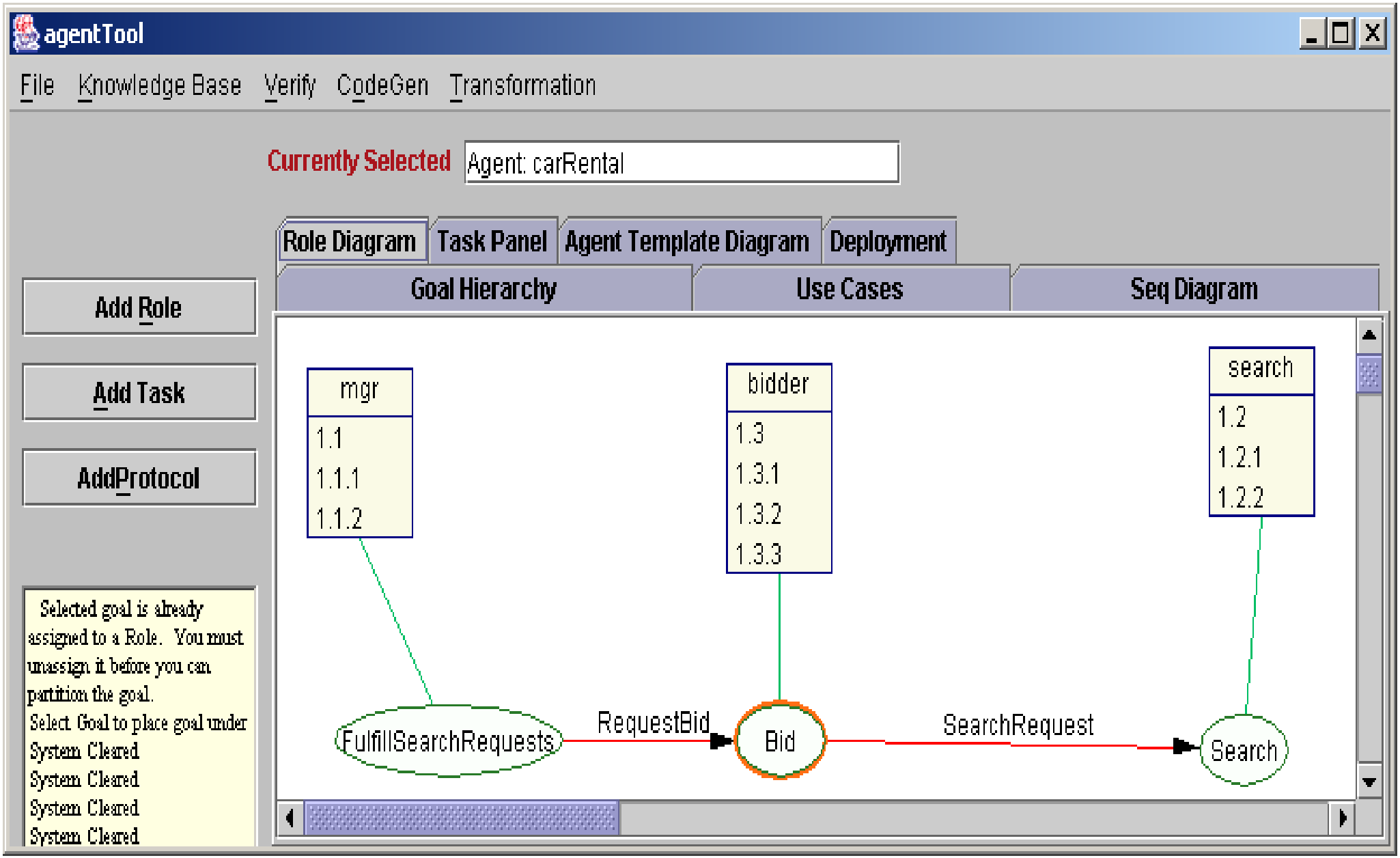
4.2. Rola *mgr:Manager* wykonuje zadanie *FulfillSearchRequests* wyboru wyszukiwarki i odbioru wyszukanej informacji

4.3. Rola *bidder:Bidder* wykonuje zadanie *Bid* uczestniczenia w licytacji wyszukiwarek

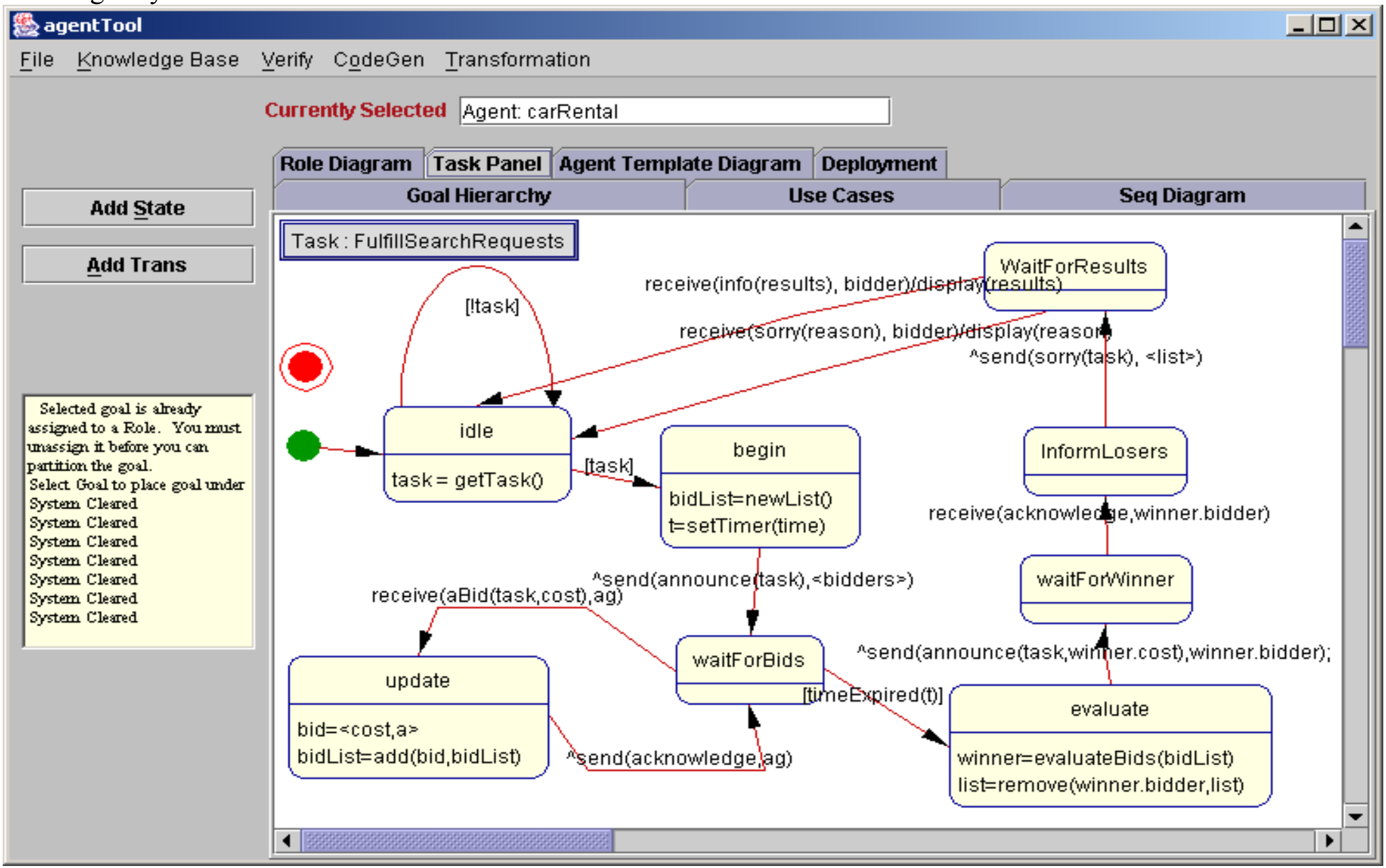
4.4. Rola *search:Search* wykonuje zadanie *Search* wyszukiwania informacji

4.5. Zadanie *FulfillSearchRequests* roli *mgr:Manager* komunikuje się z zadaniem *Bid* roli *bidder:Bidder* za pośrednictwem protokołu zewnętrznego *RequestBid*

4.6. Zadanie *Bid* roli *bidder:Bidder* komunikuje się z zadaniem *Search* roli *search:Search* za pośrednictwem zewnętrznego protokołu *SearchReguest*



Rys. 8.4. Diagram ról



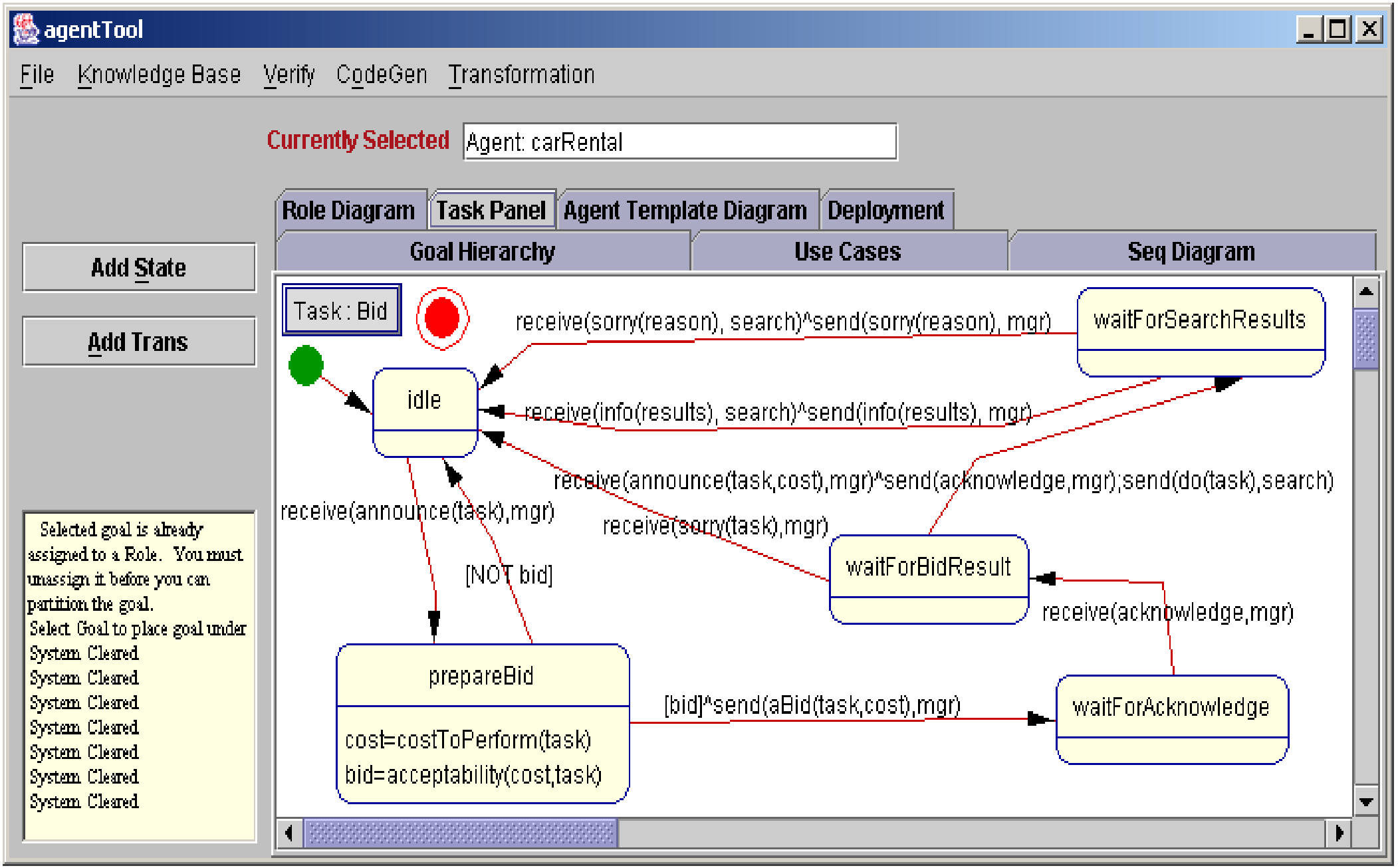
Rys. 8.5. Diagram stanu zadania *FulfillSearchRequests* roli *mgr:Manager*

Algorytm zadania *FulfilSearchRequests* roli *mgr:Manager*

1) Ze stanu początkowego przechodzi do stanu *idle* oczekiwania na zadanie
2) Po otrzymaniu poprawnego zadania przechodzi do stanu *begin,* w którym inicjuje listę ról *bidder* i ustawia czas wykonania zadania
3) Wychodzi ze stanu *begin*, wysyła komunikat zapotrzebowania na wyszukanie informacji do zadania ról *bidders (^send(announce(task,<bidders>))*
4) Przechodzi do stanu *waitForBids,* z którego wychodzi po otrzymaniu wiadomości *receive(aBid(task,cost),ag)* od zadania ról *bidders*
5) Przechodzi do stanu *update,* gdzie wstawia dane otrzymanej oferty i wychodzi ze stanu po wysłaniu komunikatu *^send(acknowledge,ag)* do zadania ról *bidders*
6) Przechodzi ponownie do stanu *waitForBid* i ponownie powtarza kroki 4 i 5
7) Po upływie wyznaczonego czasu przechodzi do stanu *evaluate*, gdzie wyznacza najlepszą rolę *bidder* i usuwa ją z listy ról *bidder*
8) Przechodzi do stanu *waitForWiner* wysyłając komunikat do zadania ról *bidder ^send(announce(task, winner.cost),winner.bidder)*
9) Po otrzymaniu komunikatu *receive(acknowledge,winner.bidder)* od zadania ról *bidders* przechodzi do stanu *InformLosers*
10) Przechodzi do stanu *wairForResults* po wysłaniu komunikatów *^send(sorry(task),<list>)* do zadania odrzuconych ról *bidders*
11) Przechodzi do stanu początkowego *idle* po otrzymaniu komunikatu *receive(info(results),bidder)/display(results)* lub komunikatu *receive(sorry(reason),bidder)/display(reason)* od zadania wybranej roli *bidder*

Algorytm zadania *Bid* roli *bidder:Bidder*

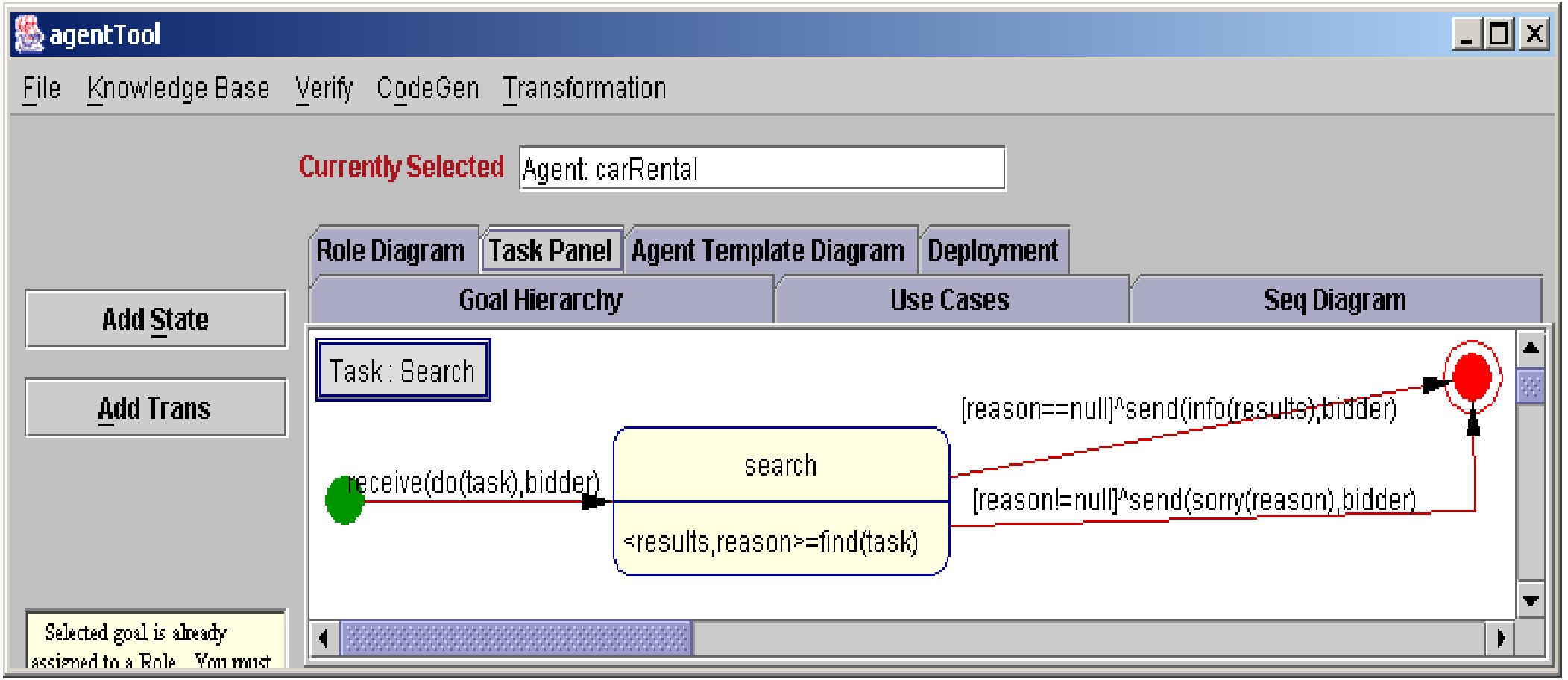
1) Po wyjściu ze stanu początkowego przechodzi do stanu oczekiwania *idle*
2) Po otrzymaniu wiadomości *receive(announce(task,<bidders>))* od zadania *FulfillRequest* roli *mgr:Manager* przechodzi do stanu *prepareBid* przygotowania oferty wyszukiwania informacji (koszt). W przypadku braku oferty przechodzi ponownie do stanu *idle*, w przeciwnym wypadku wysyła komunikat *^send(aBid(task,cost),ag)* do zadania roli *mgr:Manager*
3) Przechodzi do stanu *waitForAcknowledge,* z którego wychodzi po otrzymaniu *receive(acknowledge,ag)* od zadania roli *mgr:Manager.*
4) Przechodzi do stanu *waitForBidResuits*, z którego wychodzi po otrzymaniu wiadomości *receive(sorry(task))* od zadania roli *mgr:Manager* o odrzuceniu oferty- przechodzi wtedy do stanu *idle;*
5) Przechodzi do stanu *waitForSearchResults* po otrzymaniu komunikatu *receive(announce(task,.cost),mgr)* od zadania roli *mgr:Manager* i wysłaniu do niej komunikatu *^send(acknowledge,mgr)* oraz komunikatu *^send(do(task),search)* do zadania *Search* roli *search:Search* zlecającego jej wyszukiwanie informacji
6) Przechodzi do stanu *idle* po otrzymaniu od zadania roli *search:Search* jednego z dwóch komunikatów: *receive(info(results),search) lub receive(sorry(reason),search)* i wysłaniu do zadania roli *mgr:Manager* jednego z dwóch komunikatów *^send(info(results),mgr)* lub *^send(sorry(reason),mgr)*



Rys.8.6. Diagram stanu zadania *Bid* roli *bidder:Bidder*

Algorytm zadania *Search* roli *search:Search*

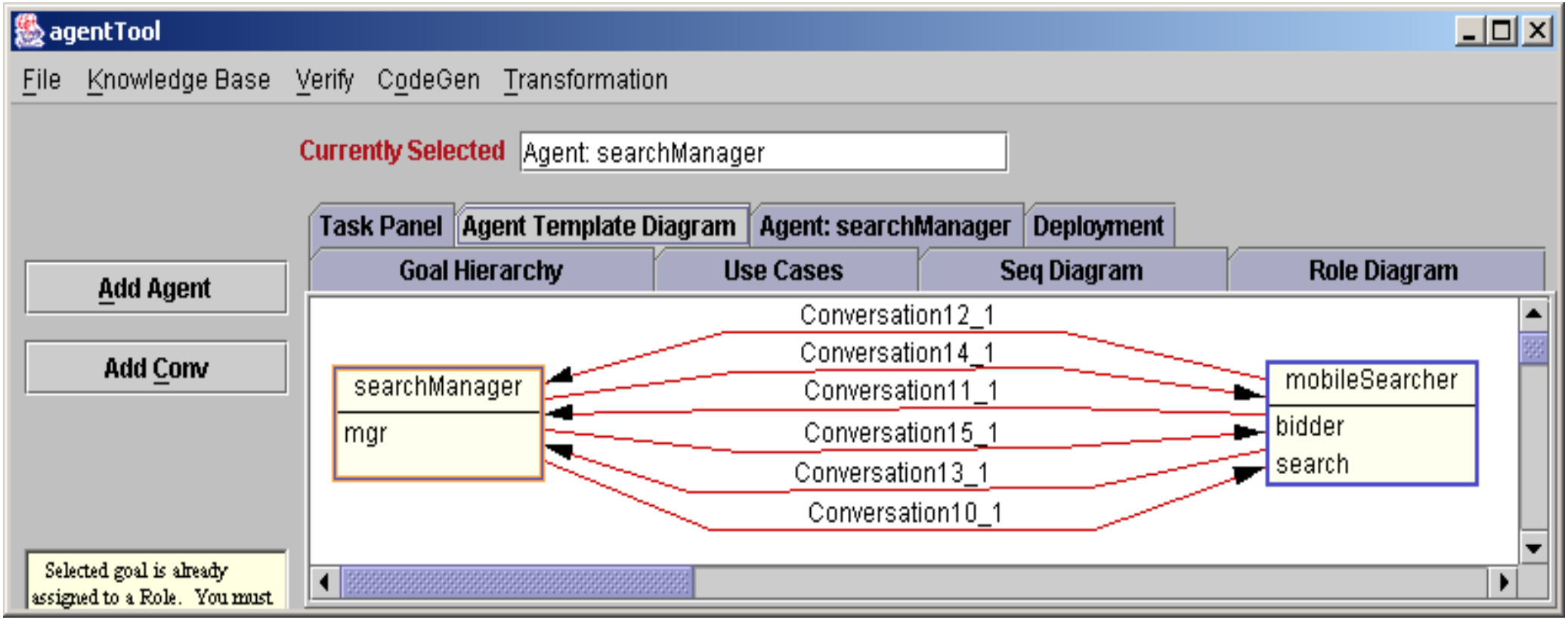
1) Wychodzi ze stanu początkowego po otrzymaniu komunikatu *receive(do(task),bidder)* od zadania roli *bidder:Bidder* i przechodzi do stanu *search*, w którym gromadzi wyszukaną informację
2) Wychodzi ze stanu wyszukiwania po wykonaniu zadania wyszukiwania i wysyła jeden z dwóch komunikatów *^send(info(results),bidder)* lub *^send(sory(reason),bidder)* do zadania roli *bidder:Bidder*
3) Kończy zadanie *Search* przechodząc do stanu zakończenia zadania



Rys. 8.7. Diagram stanu zadania *Search* roli *search:Search*

II. Projektowanie

1) Diagram agentów



Rys. 8.8. Diagram agentów i konwersacji

Diagram powstał na podstawie diagramów ról i zadań wygenerowanych podczas analizy

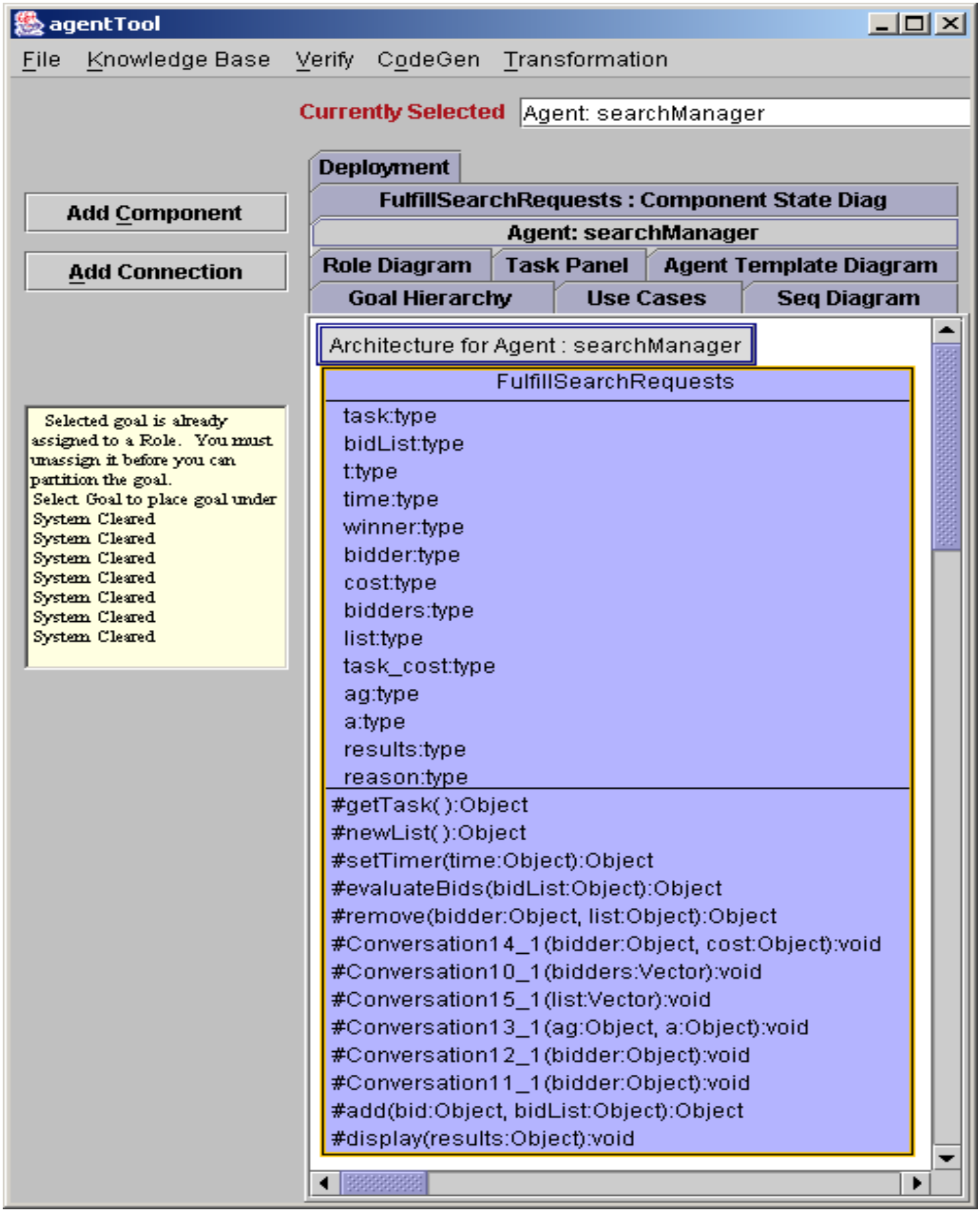
1.1 Rola *mgr:Manager* wraz z zadaniem *FulfillSearchRequest* została przydzielona agentowi *Manager*

1.2 Role *bidder:Bidder* oraz *search:Search* wraz z zadaniami *Bid* oraz *Search* zostały przydzielone agentowi *mobileSearcher*

1.3 Konwersacje 10_1 do 15_1 realizują protokół komunikacji między agentami, wynikający z protokołów *RequestBid* oraz *SearchRequest,* pozwalających na komunikację między zadaniami *FulfillSearchRequests* roli *mgr:Manager*, *Bid* roli *bidder:Bidder* oraz *Search roli search:Search.*

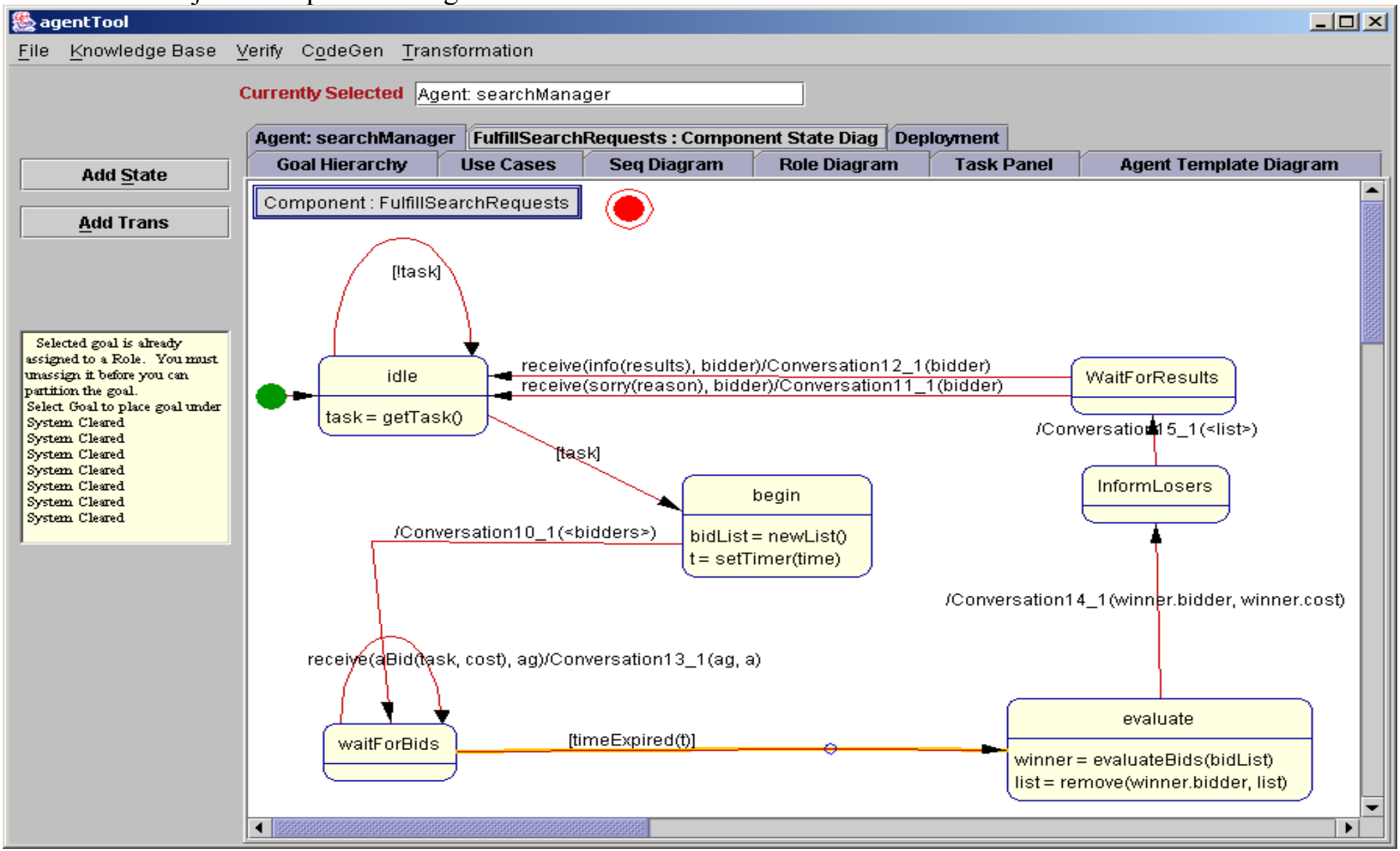
2. Diagram komponentów agenta *searchManager*

Komponent agenta *searchManager* realizuje zadanie *FulfillSearchRequest* roli *mgr:Manager*



Rys. 8.9. Komponent *FulfillSearchRequests*

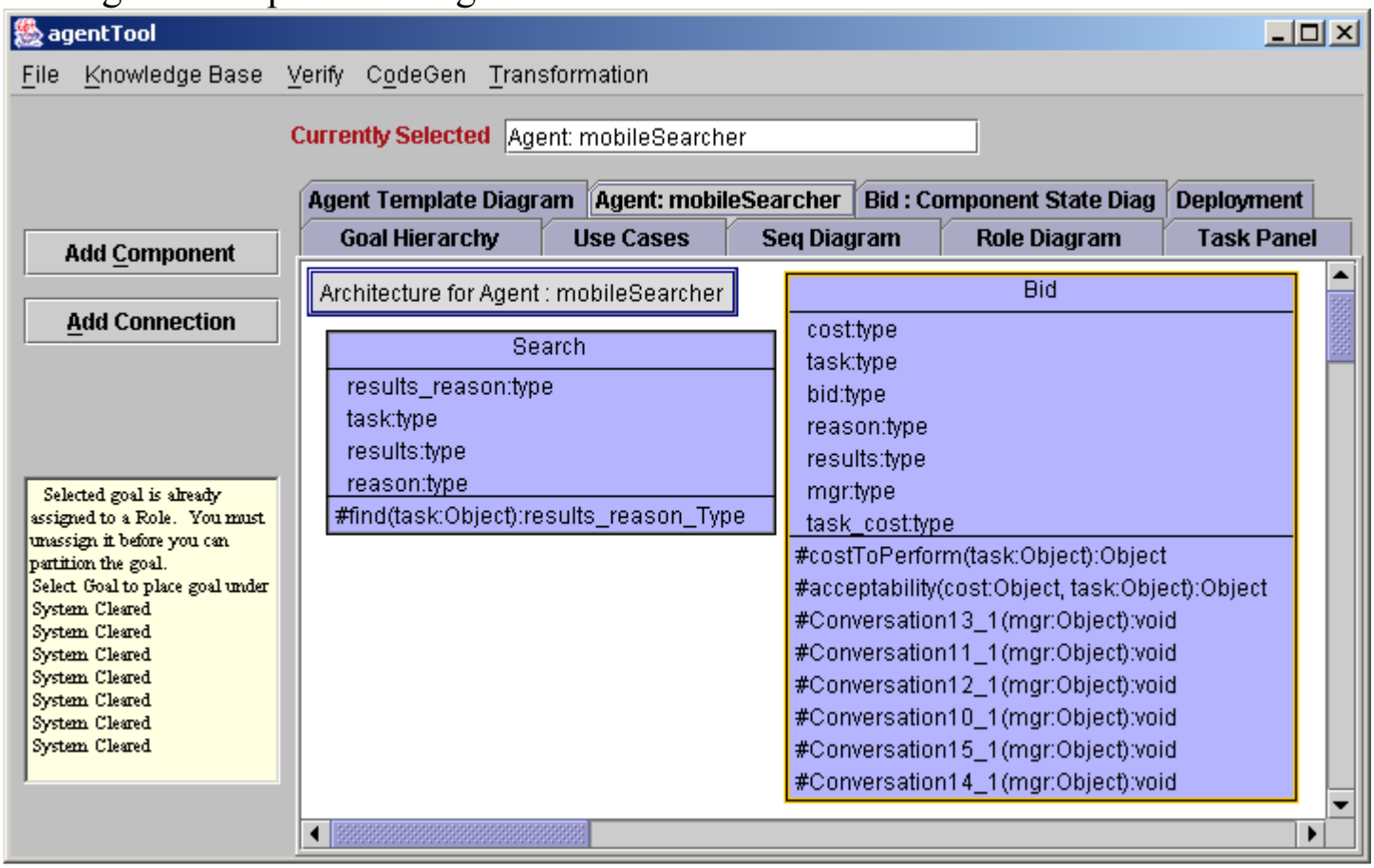
Diagram stanu komponentu *FulfillSearchRequests*- na diagramie podano konwersacje realizujące nadawanie lub odbiór informacji od komponentów agenta *mobileSearch*



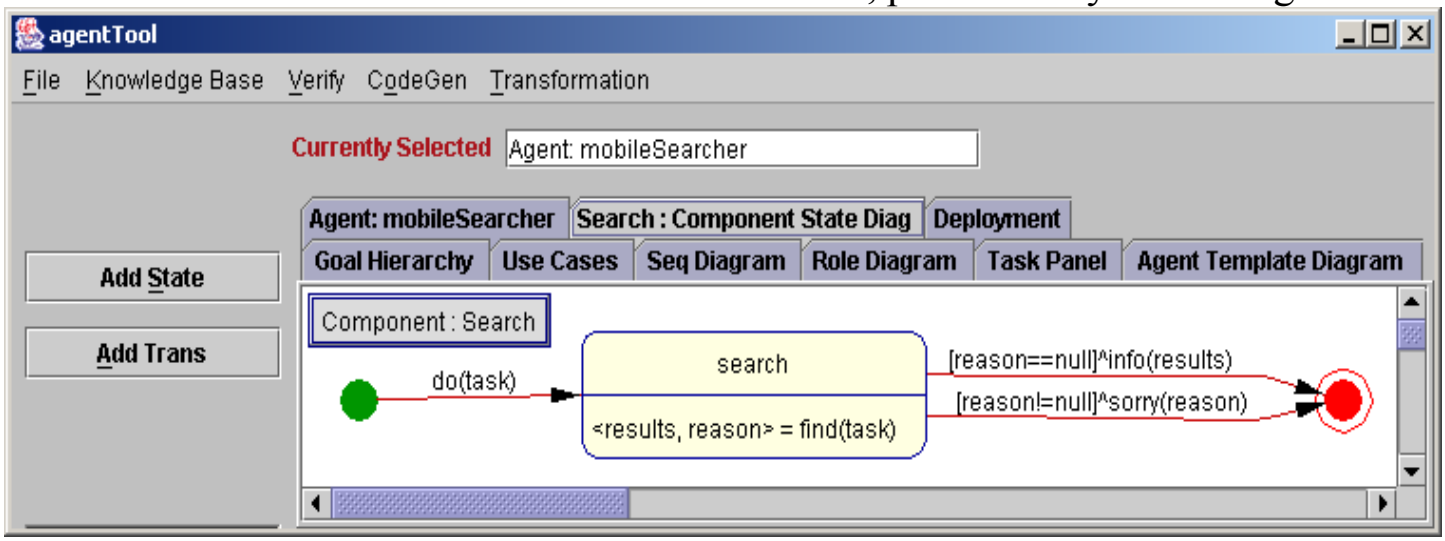
Rys.8.10. Diagram stanu komponentu *FulfillSearchRequest* agenta *searchManager* (patrz zadanie z rys. 8.5)

Conversation10_1 - *^send(announce(task),<bidders>)*
Conversation13_1 - *receive(aBid(task,cost),ag), ^send(acknowledge, ag)*
Conversation14_1 - *^send(announce(task, winner.cost), winner.bidder), receive(acknowledge,winner.bidder)*
Conversation15_1 - *^send(sorry(task),<list>)*
Conversation11_1 - *receive(sorry(reason),bidder)*
Conversation12_1 - *receive(info(results),bidder)*

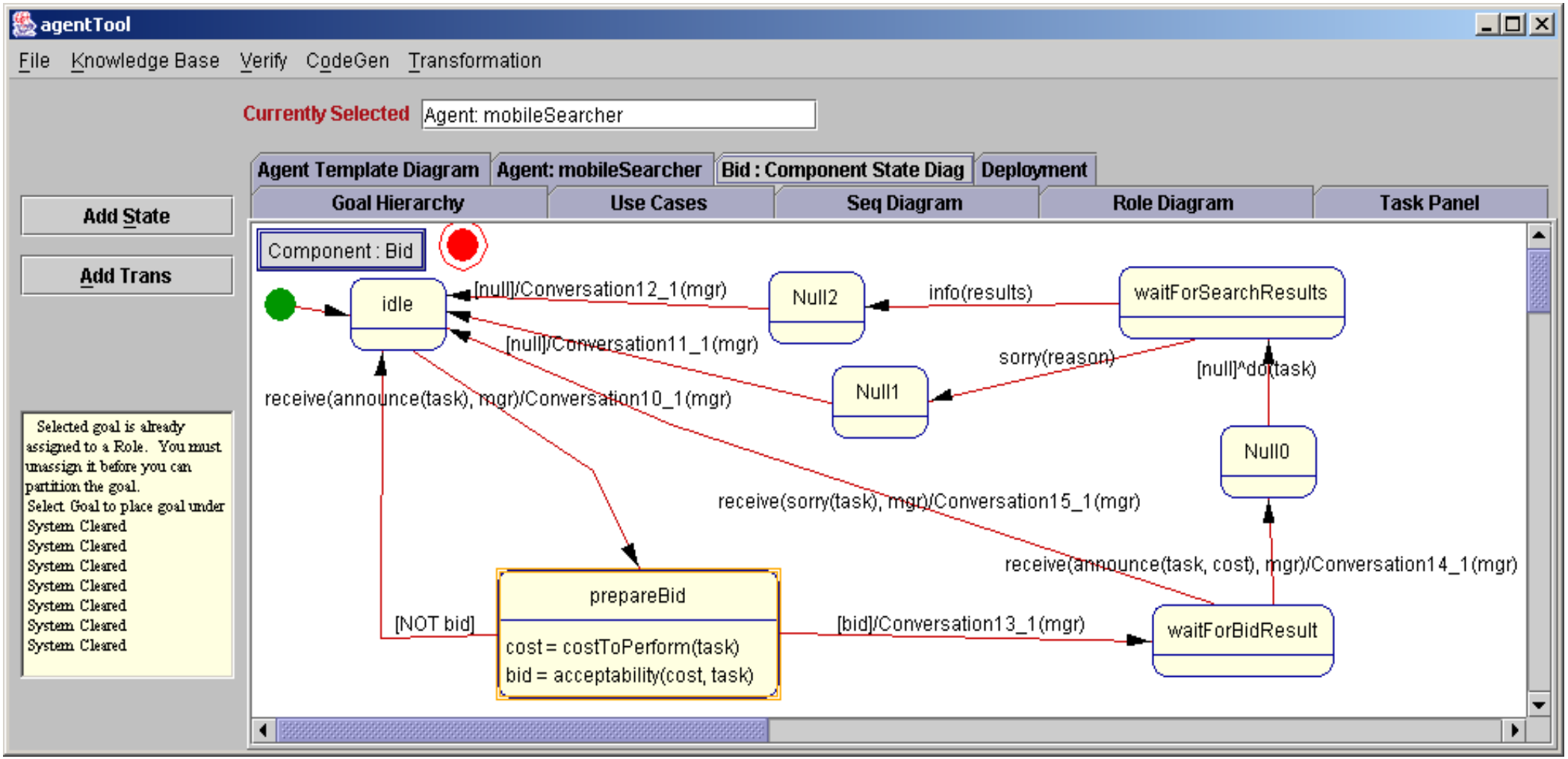
3. Diagram komponentów agenta *mobileSearch*



Rys.8.11. Komponenety agenta *mobileSearch* - wywodzą się z zadania *Bid* roli *bidder:Bidder* i zadania *Search* roli *search:Search*, przedzielonych temu agentowi.



Rys.8.12. Diagram stanu komponentu *Search*



Rys.8.13. Diagram stanu komponentu *Bid*

Conversation10_1 - *receive(announce(task),mgr)*

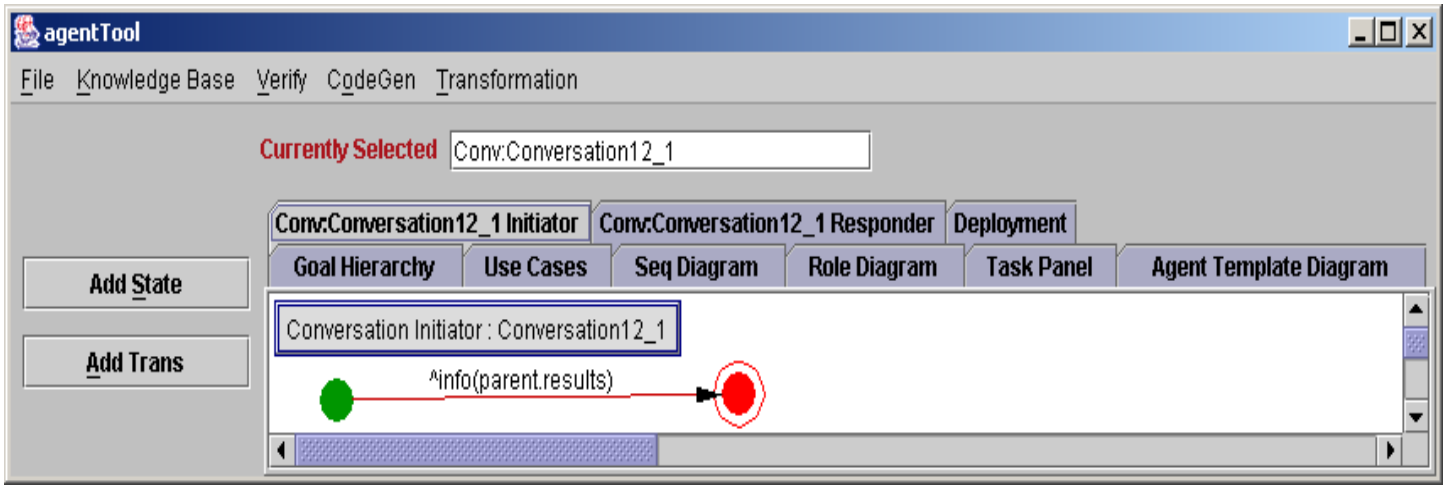
Conversation13_1 - *^send(aBid(task,cost),mgr) , receive (acknowledge,mgr)*

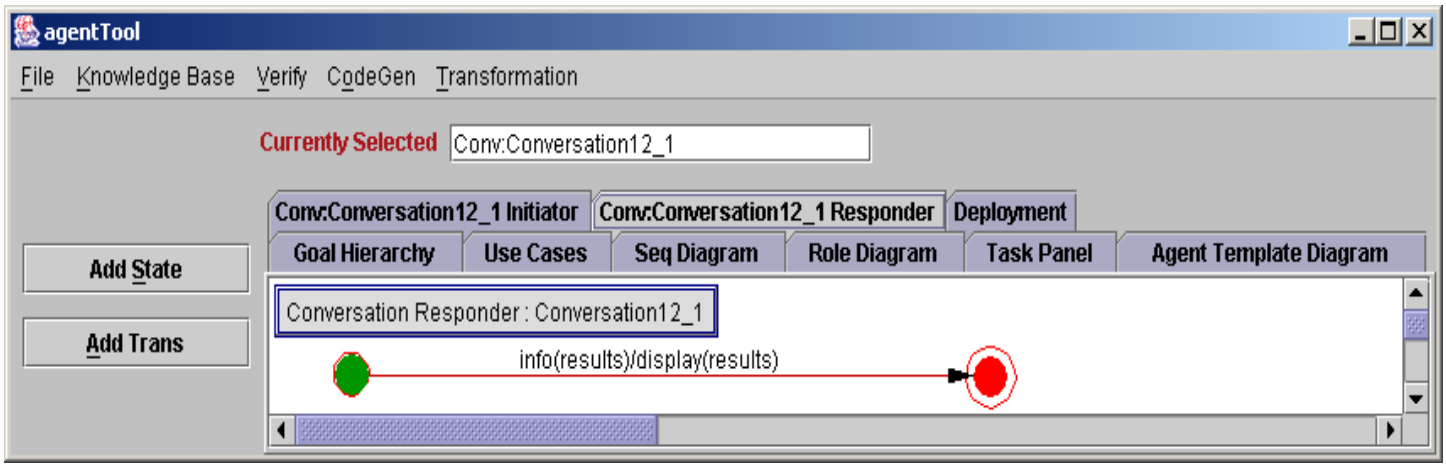
Conversation14_1 - *receive(announce(task, cost), mgr), ^send(acknowledge,mgr)*

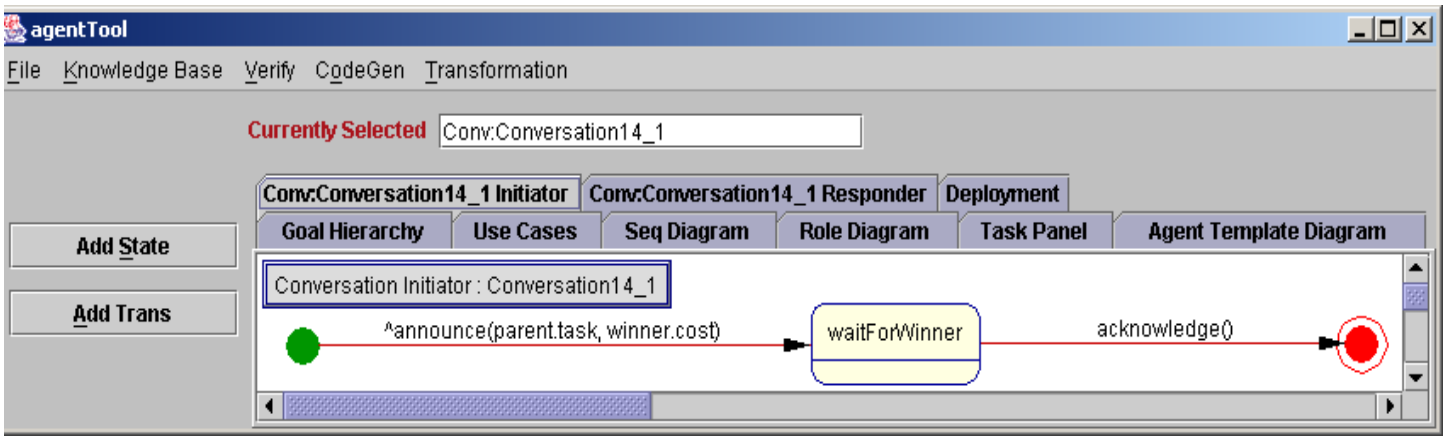
Conversation15_1 - *receive(sorry(task),mgr)*

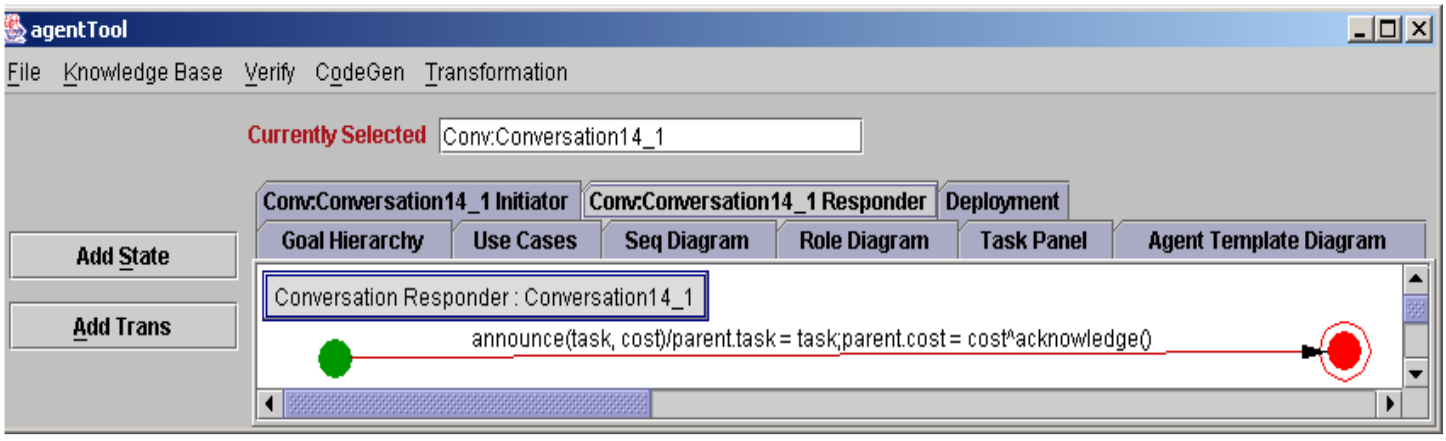
Conversation11_1 - *^send(sorry(reason),mgr)*

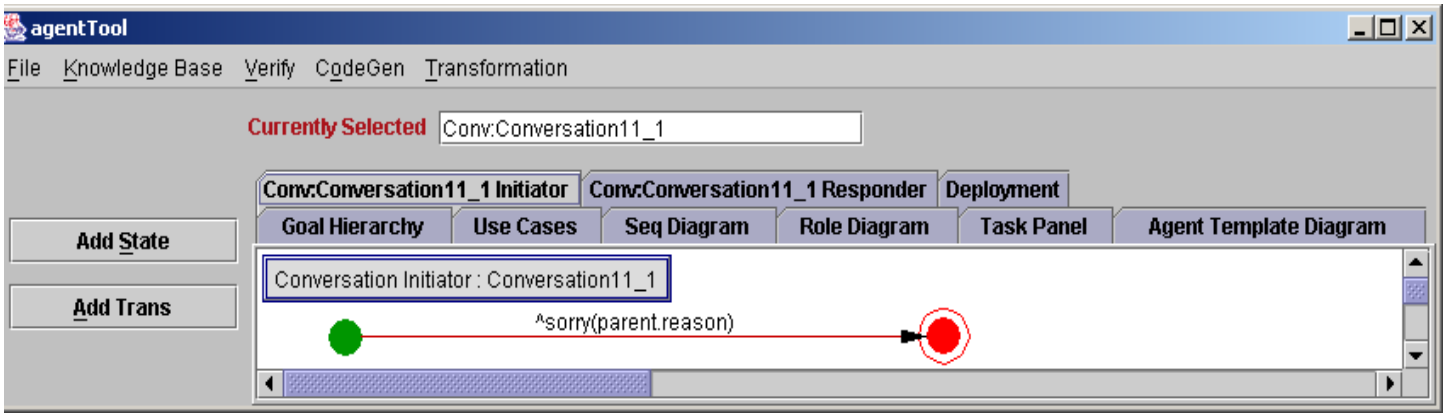
Conversation12_1 – *^send(info(results),mgr)*

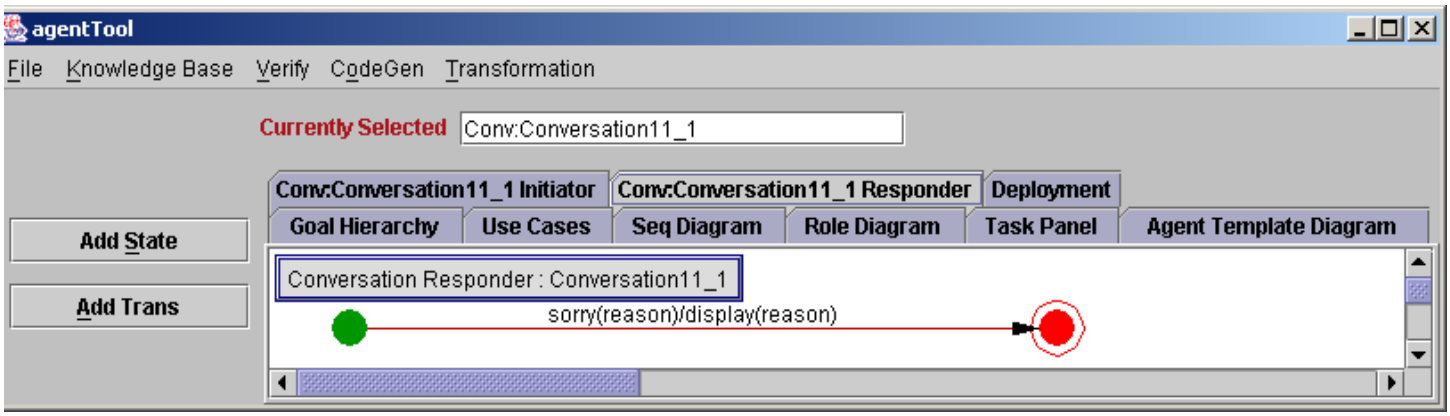

agentTool
File Knowledge Base Verify CodeGen Transformation
Currently Selected
Conv:Conversation12_1
Conv:Conversation12_1 Initiator
Conv:Conversation12_1 Responder
Deployment
Goal Hierarchy
Use Cases
Seq Diagram
Role Diagram
Task Panel
Agent Template Diagram
Add State
Add Trans
Conversation Initiator : Conversation12_1
^info(parent.results)

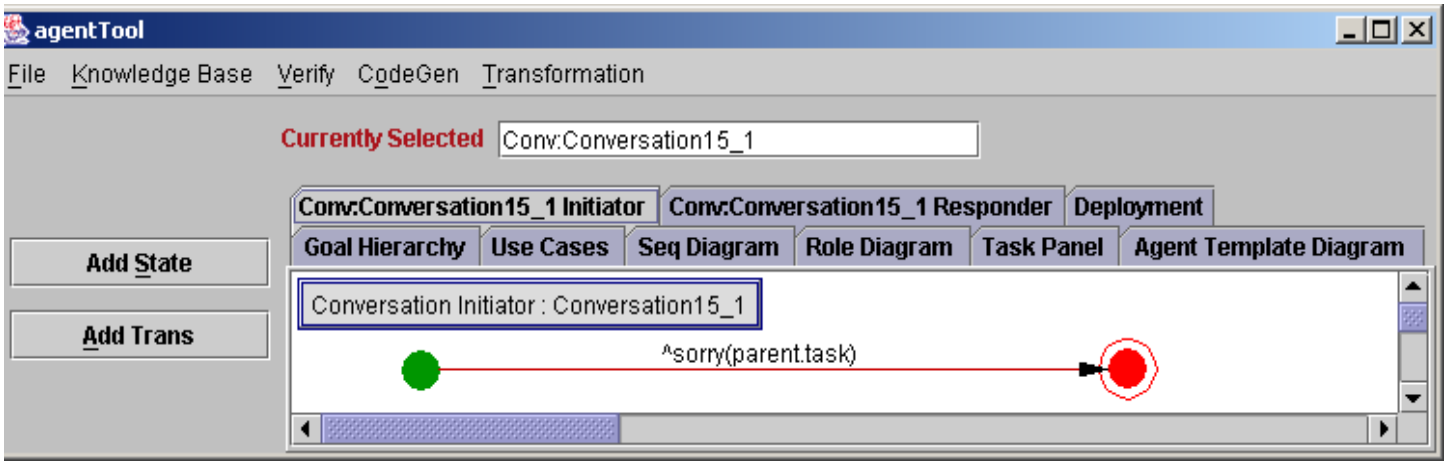

agentTool
File Knowledge Base Verify CodeGen Transformation
Currently Selected
Conv:Conversation12_1
Conv:Conversation12_1 Initiator
Conv:Conversation12_1 Responder
Deployment
Goal Hierarchy
Use Cases
Seq Diagram
Role Diagram
Task Panel
Agent Template Diagram
Add State
Add Trans
Conversation Responder : Conversation12_1
info(results)/display(results)

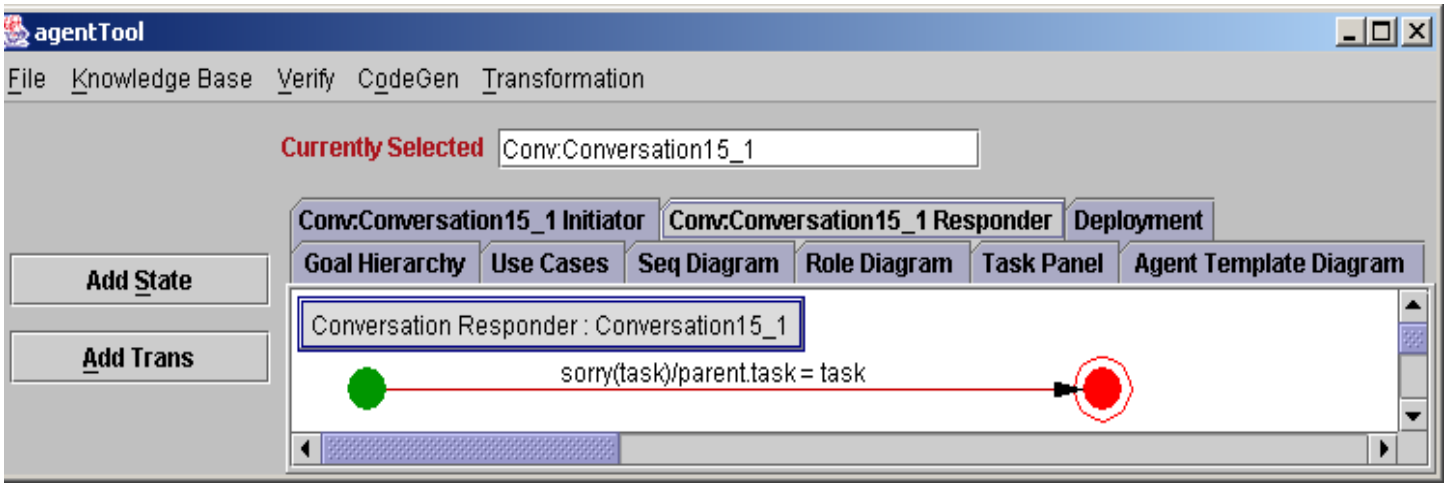

agentTool
File Knowledge Base Verify CodeGen Transformation
Currently Selected
Conv:Conversation14_1
Conv:Conversation14_1 Initiator
Conv:Conversation14_1 Responder
Deployment
Goal Hierarchy
Use Cases
Seq Diagram
Role Diagram
Task Panel
Agent Template Diagram
Add State
Add Trans
Conversation Initiator : Conversation14_1
^announce(parent.task, winner.cost)
waitForWinner
acknowledge()

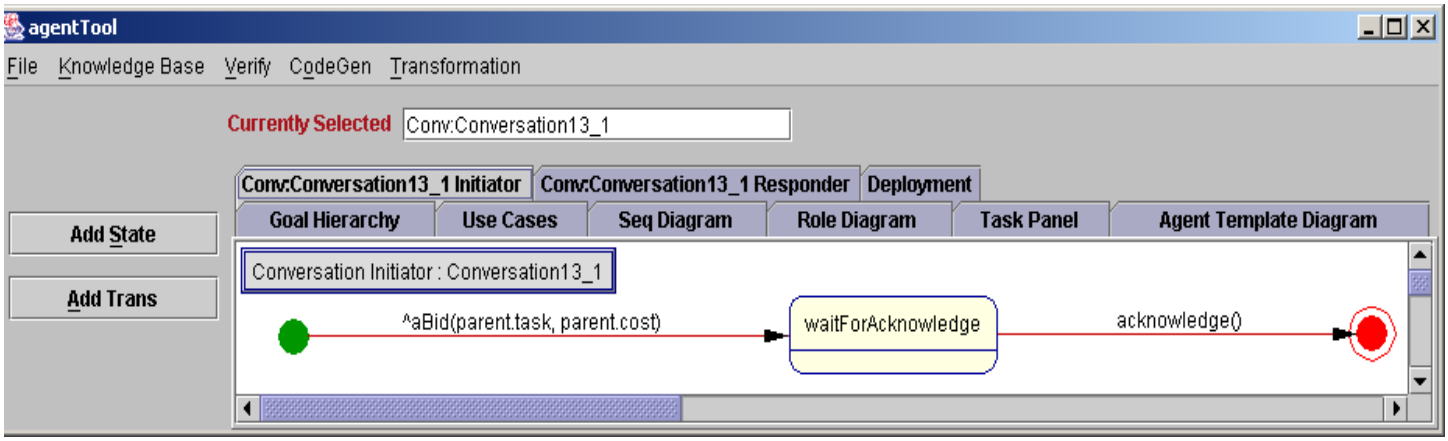

agentTool
File Knowledge Base Verify CodeGen Transformation
Currently Selected
Conv:Conversation14_1
Conv:Conversation14_1 Initiator
Conv:Conversation14_1 Responder
Deployment
Goal Hierarchy
Use Cases
Seq Diagram
Role Diagram
Task Panel
Agent Template Diagram
Add State
Add Trans
Conversation Responder : Conversation14_1
announce(task, cost)/parent.task = task;parent.cost = cost^acknowledge()

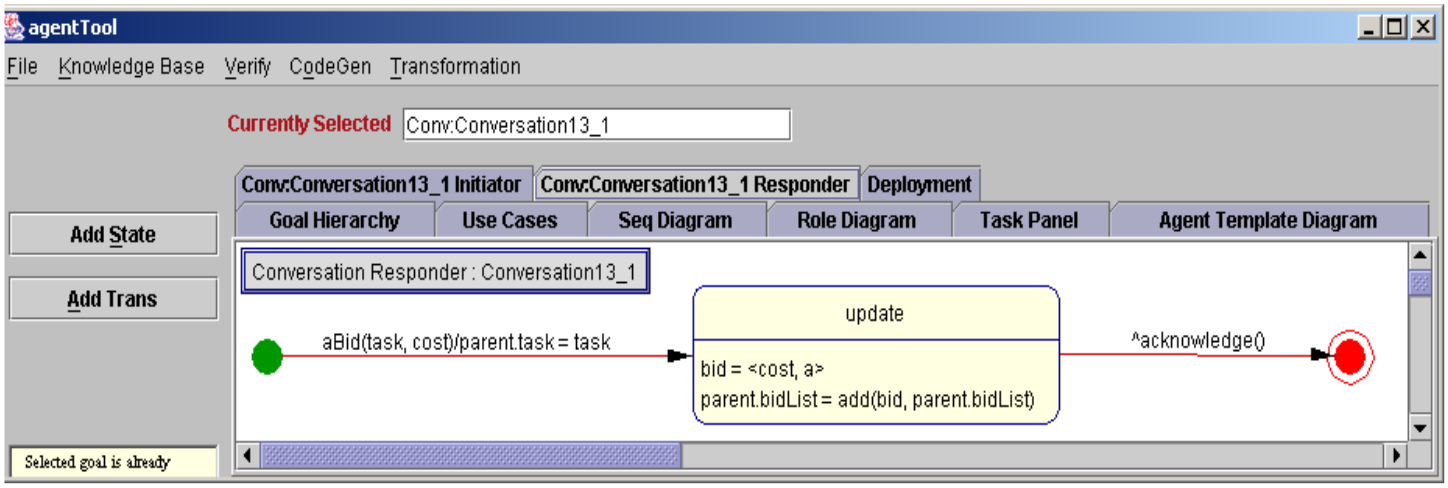

agentTool
File Knowledge Base Verify CodeGen Transformation
Currently Selected
Conv:Conversation11_1
Conv:Conversation11_1 Initiator
Conv:Conversation11_1 Responder
Deployment
Goal Hierarchy
Use Cases
Seq Diagram
Role Diagram
Task Panel
Agent Template Diagram
Add State
Add Trans
Conversation Initiator : Conversation11_1
^sorry(parent.reason)

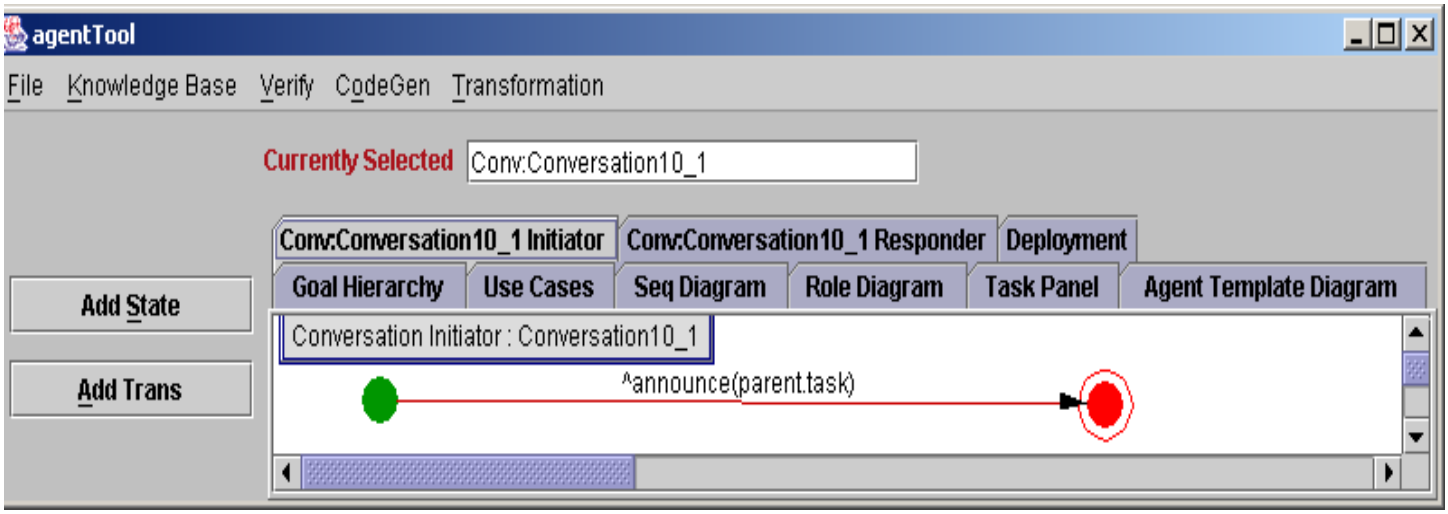

agentTool
File Knowledge Base Verify CodeGen Transformation
Currently Selected
Conv:Conversation11_1
Conv:Conversation11_1 Initiator
Conv:Conversation11_1 Responder
Deployment
Goal Hierarchy
Use Cases
Seq Diagram
Role Diagram
Task Panel
Agent Template Diagram
Add State
Add Trans
Conversation Responder : Conversation11_1
sorry(reason)/display(reason)

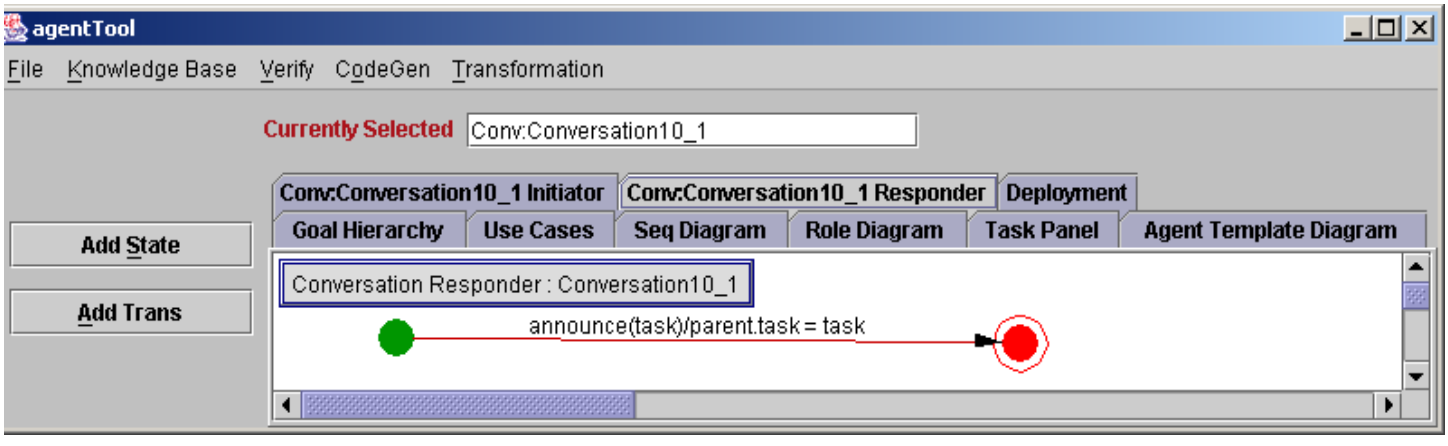

agentTool
File Knowledge Base Verify CodeGen Transformation
Currently Selected
Conv:Conversation15_1
Conv:Conversation15_1 Initiator
Conv:Conversation15_1 Responder
Deployment
Goal Hierarchy
Use Cases
Seq Diagram
Role Diagram
Task Panel
Agent Template Diagram
Add State
Add Trans
Conversation Initiator : Conversation15_1
^sorry(parent.task)


agentTool
File Knowledge Base Verify CodeGen Transformation
Currently Selected
Conv:Conversation15_1
Conv:Conversation15_1 Initiator
Conv:Conversation15_1 Responder
Deployment
Goal Hierarchy
Use Cases
Seq Diagram
Role Diagram
Task Panel
Agent Template Diagram
Add State
Add Trans
Conversation Responder : Conversation15_1
sorry(task)/parent.task = task


agentTool
File Knowledge Base Verify CodeGen Transformation
Currently Selected
Conv:Conversation13_1
Conv:Conversation13_1 Initiator
Conv:Conversation13_1 Responder
Deployment
Goal Hierarchy
Use Cases
Seq Diagram
Role Diagram
Task Panel
Agent Template Diagram
Add State
Add Trans
Conversation Initiator : Conversation13_1
^aBid(parent.task, parent.cost)
waitForAcknowledge
acknowledge()


agentTool
File Knowledge Base Verify CodeGen Transformation
Currently Selected
Conv:Conversation13_1
Conv:Conversation13_1 Initiator
Conv:Conversation13_1 Responder
Deployment
Goal Hierarchy
Use Cases
Seq Diagram
Role Diagram
Task Panel
Agent Template Diagram
Add State
Add Trans
Conversation Responder : Conversation13_1
update
aBid(task, cost)/parent.task = task
bid = <cost, a>
parent.bidList = add(bid, parent.bidList)
^acknowledge()
Selected goal is already


agentTool
File Knowledge Base Verify CodeGen Transformation
Currently Selected
Conv:Conversation10_1
Conv:Conversation10_1 Initiator
Conv:Conversation10_1 Responder
Deployment
Goal Hierarchy
Use Cases
Seq Diagram
Role Diagram
Task Panel
Agent Template Diagram
Add State
Add Trans
Conversation Initiator : Conversation10_1
^announce(parent.task)


agentTool
File Knowledge Base Verify CodeGen Transformation
Currently Selected
Conv:Conversation10_1
Conv:Conversation10_1 Initiator
Conv:Conversation10_1 Responder
Deployment
Goal Hierarchy
Use Cases
Seq Diagram
Role Diagram
Task Panel
Agent Template Diagram
Add State
Add Trans
Conversation Responder : Conversation10_1
announce(task)/parent.task = task