JESS

Tomasz Rybak
rybak@ii.pb.bialystok.pl

Applied Systems Division
Software Departament
Faculty of Computer Science
Bialystok Technical University

Table of content

JESS

- Strona http://www.jessrules.com/
- Szkieletowy system ekspertowy
- Język implementacji Java
- Język opisu reguł kompatybilny z CLIPS
- Używa algorytmy Rete
  http://herzberg.ca.sandia.gov/jess/docs/70/rete.html
- Wersja 7.0 umożliwia wnioskowanie w przód i wstecz
- Licencja komercyjna i bezpłatna, akademicka
- Wersja próbna działająca 30 dni

Krótki opis

- Możliwość dostępu do obiektów systemu korzystając z języka Java
- Możliwość tworzenia UI korzystając z biblioteki AWT
- Możliwość rozszerzania funkcjonalności wstawkami w języku Java
- Dokumentacja http://www.jessrules.com/jess/docs/index.shtml
- Wiki: http://www.jessrules.com/jesswiki/view

Struktura katalogu

Program rozprowadzany jako archiwum zip

bin	skrypty służące do uruchownienia programu w Uniksach i pod Windows
docs	dokumentacja
eclipse	wtyczki do Eclipse
examples	przykładowe programy
lib	implementaja systemu, pliki jar z kodem

Przykłady

- Przykłady zapisane są zarówno w języku CLIPS jak i w kodzie XML
- Dylemat więźnia
- Bierki
- Małpa i banan
- Rozwiązywanie równań literowych

Współpraca z Eclipse

- Eclipse to open-source IDE, napisane w języku Java
- Strona http://www.eclipse.org/
- Umożliwia pisane programów w językach C++ i Java
- Można je rozszerzać dodając wtyczki
- Istnieją wtyczki do Perla, Pythona, PHP, Lispa i Scheme i innych

Konfiguracja wtyczek

- Pliki zip z wtyczkami znajdują się w katalogu eclipse/
- Należy je wszystkie rozpakować do katalogu w którym
- W tym katalogu tworzymi plik .eclipseextension
- Następnie Help− >Software Updates− >Manage Configuration
- Wyiramy Add an Extension Location
- Dodajemy katalog do którego rozpakowaliśmy wtyczki

Plik .eclipseextension

```
id=org.eclipse.platform
name=Eclipse Platform
version=3.2.1
```

3.2.1 to wersja Eclipse z której korzystamy

Wtyczki

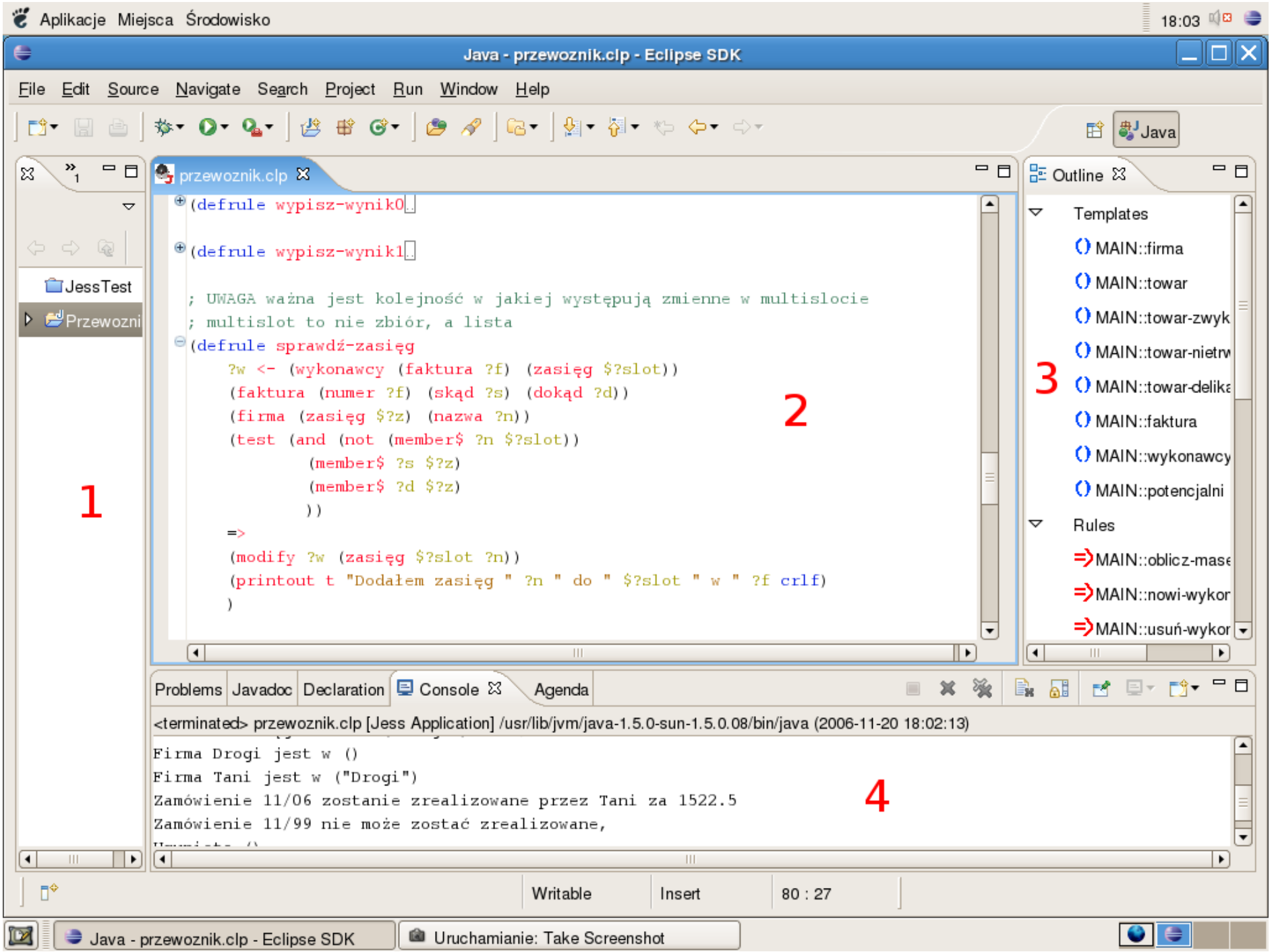
gov.sandia.jess_7.0.0.zip łączące Eclipse i Jess

gov.sandia.jess.debug_7.0.0.zip używana przy wyszukiwaniu błędów

gov.sandia.jess.editor_7.0.0.zip używana przez edytor kodu Jess

gov.sandia.jess.feature_7.0.0.zip

gov.sandia.jess.reteview_7.0.0.zip używana do wyświetlania sieci Rete

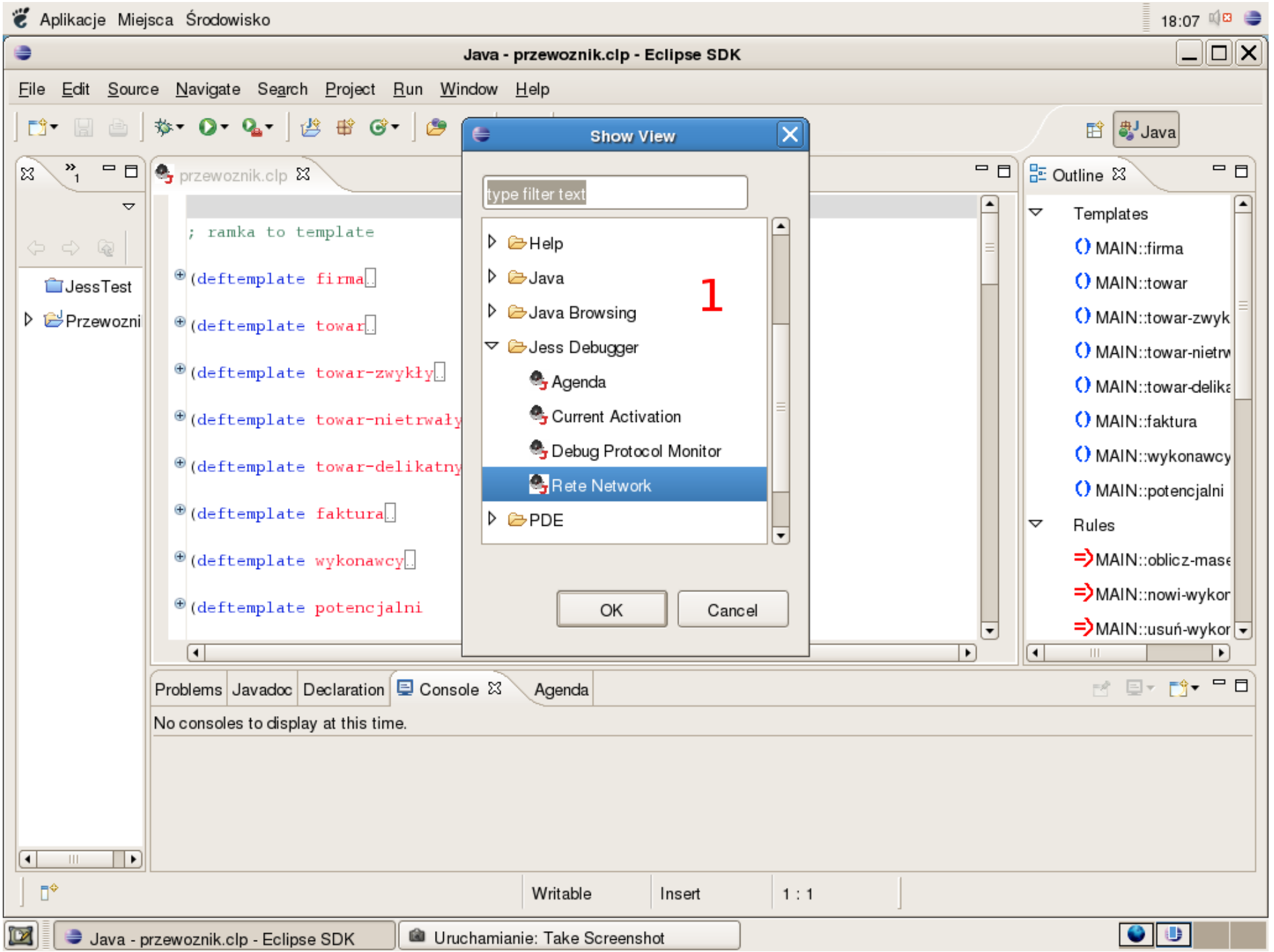

Aplikacje Miejsca Środowisko
18:03
Java - przewoznik.clp - Eclipse SDK
File Edit Source Navigate Search Project Run Window Help
Java
JessTest
Przewozni
1
przewoznik.clp
(defrule wypisz-wynik0
(defrule wypisz-wynik1
; UWAGA ważna jest kolejność w jakiej występują zmienne w multislocie
; multislot to nie zbiór, a lista
(defrule sprawdź-zasięg
?w <- (wykonawcy (faktura ?f) (zasięg $?slot))
(faktura (numer ?f) (skąd ?s) (dokąd ?d))
(firma (zasięg $?z) (nazwa ?n))
(test (and (not (member$?n $?slot))
(member$?s $?z)
(member$?d $?z)
))
=>
(modify ?w (zasięg $?slot ?n))
(printout t "Dodałem zasięg " ?n " do " $?slot " w " ?f crlf)
)
2
Outline
Templates
MAIN::firma
MAIN::towar
MAIN::towar-zwyk
MAIN::towar-nietrw
MAIN::towar-delika
MAIN::faktura
MAIN::wykonawcy
MAIN::potencjalni
Rules
MAIN::oblicz-mase
MAIN::nowi-wykor
MAIN::usuń-wykor
3
Problems Javadoc Declaration Console Agenda
<terminated> przewoznik.clp [Jess Application] /usr/lib/jvm/java-1.5.0-sun-1.5.0.08/bin/java (2006-11-20 18:02:13)
Firma Drogi jest w ()
Firma Tani jest w ("Drogi")
Zamówienie 11/06 zostanie zrealizowane przez Tani za 1522.5
Zamówienie 11/99 nie może zostać zrealizowane,
4
Writable
Insert
80 : 27
Java - przewoznik.clp - Eclipse SDK
Uruchamianie: Take Screenshot

Widok okna Eclipse

1. Widok wszystkich istniejących projektów, zarówno otwartych jak i zamkniętych
2. Okno edytora
3. Widok wszyskich istniejących obiektów systemu Jess
4. Konsola z wynikiem działania systemu

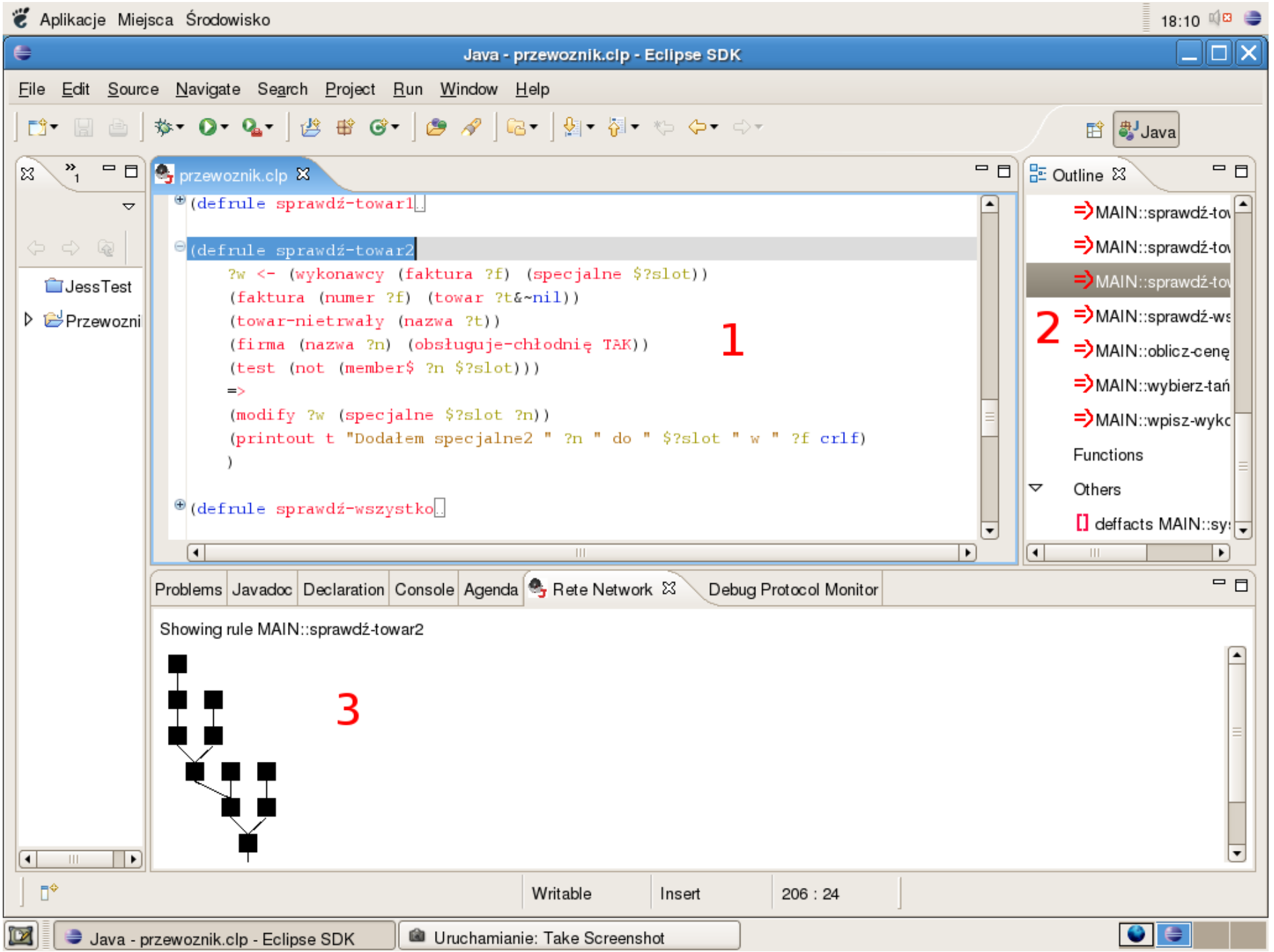
Konfiguracja Eclipse

- Aby móc wyświetlić sieć Rete, potrzebne jest rozszerzenie Graphical Editing Framework (GEF)
- Adres http://www.eclipse.org/gef/
- Eclipse w wersji 3.2 jest połączony z kilkoma najpopularniejszymi rozszerzeniami, w tym z GEF
- To połączenie nazywane jest Callisto
- Adres http://www.eclipse.org/callisto/


Aplikacje Miejsca Środowisko
18:07
Java - przewoznik.clp - Eclipse SDK
File Edit Source Navigate Search Project Run Window Help
Java
przewoznik.clp
; ramka to template
(deftemplate firma
(deftemplate towar
(deftemplate towar-zwykły
(deftemplate towar-nietrwały
(deftemplate towar-delikatny
(deftemplate faktura
(deftemplate wykonawcy
(deftemplate potencjalni
JessTest
Przewozni
Show View
type filter text
Help
Java
Java Browsing
Jess Debugger
Agenda
Current Activation
Debug Protocol Monitor
Rete Network
PDE
1
OK
Cancel
Outline
Templates
MAIN::firma
MAIN::towar
MAIN::towar-zwyk
MAIN::towar-nietrw
MAIN::towar-delika
MAIN::faktura
MAIN::wykonawcy
MAIN::potencjalni
Rules
MAIN::oblicz-mase
MAIN::nowi-wykor
MAIN::usuń-wykor
Problems Javadoc Declaration Console Agenda
No consoles to display at this time.
Writable Insert 1 : 1
Java - przewoznik.clp - Eclipse SDK
Uruchamianie: Take Screenshot

Widok okna Eclipse

1. Okno wyboru widoku sieci Rete (Window− >View− >Other)


Aplikacje Miejsca Środowisko
18:10
Java - przewoznik.clp - Eclipse SDK
File Edit Source Navigate Search Project Run Window Help
Java
przewoznik.clp
(defrule sprawdź-towar1
(defrule sprawdź-towar2
?w <- (wykonawcy (faktura ?f) (specjalne $?slot))
(faktura (numer ?f) (towar ?t&~nil))
(towar-nietrwały (nazwa ?t))
(firma (nazwa ?n) (obsługuje-chłodnię TAK))
(test (not (member$?n $?slot)))
=>
(modify ?w (specjalne $?slot ?n))
(printout t "Dodałem specjalne2 " ?n " do " $?slot " w " ?f crlf)
)
(defrule sprawdź-wszystko
1
Outline
MAIN::sprawdź-to
MAIN::sprawdź-to
MAIN::sprawdź-to
MAIN::sprawdź-ws
MAIN::oblicz-cenę
MAIN::wybierz-tań
MAIN::wpisz-wykc
Functions
Others
deffacts MAIN::sy
2
JessTest
Przewozni
Problems Javadoc Declaration Console Agenda Rete Network Debug Protocol Monitor
Showing rule MAIN::sprawdź-towar2
3
Writable
Insert
206 : 24
Java - przewoznik.clp - Eclipse SDK
Uruchamianie: Take Screenshot

Widok okna Eclipse

1. Okno edytora z zaznaczoną regułą której sieć Rete jest wyświetlana
2. Ta sama reguła podświetlona w oknie widoku obiektów systemu Jess
3. Sieć Rete dla tej reguły

System przewoźnik

- System na którego przykładzie zostanie przedstawiony system Jess
- Istnieje wiele firm przewozowych
- Istnieją różne typy towarów
- Istnieje wielu klientów którzy chcą przewozić towary między miastami
- Dla każdego klienta należy wybrać najtańszego przewoźnika spośród tych, którzy są w stanie podołać zleceniu

Wzorce

- Definiują typy z których korzystamy w systemie
- Wzorzec (ang. template) nazywany jest często ramą (ang. frame)
- Wzorce mogą dziedziczyć z innych wzorców
- Wzorzec składa się z nazwanych slotów
- Każdy slot może mieć typ, i przechowuje jedną wartość
- Jeśli chcemy przechowywać więcej wartości, zamiast slotu używamy multislotukonieczny jest
- Slot nie może przechowywać odwołania do ram ani faktów
- Aby opisać powiązania między obiektami, posługujemy się techniką znaną z baz danych, korzystając z identyfikatorów

Tworzenie ram

Jess

```
(deftemplate firma
(slot nazwa (type string))
(slot cena-przewozu (type float))
(slot cena-załadunku (type float))
(multislot zasięg (type string))
(multislot flota (type string))
(slot obsługuje-specjalne)
(slot obsługuje-chłodnię)
(slot ładowność (type float))
)
```

Tworzenie ram

Java

```
Rete r = new Rete();
Deftemplate dt = new Deftemplate("firma",
"Komentarz", r);
dt.addSlot("nazwa", null, "STRING");
dt.addSlot("cena-przewozu", null, "FLOAT");
dt.addSlot("cena-załadunku", null, "FLOAT");
dt.addMultiSlot("zasięg", null, "STRING");
dt.addMultiSlot("flota", null, "STRING");
dt.addSlot("obsługuje-specjalne)", null, "ANY");
dt.addSlot("obsługuje-chłodnię)", null, "ANY");
dt.addSlot("ładowność", new Value(0, RU.FLOAT), "FLOAT");
r.addDeftemplate(dt);
```

Asercje

- Zwykle są to nazwane ciągi (listy) wartości
- Nie występują w Jess jako oddzielny typ
- Odpowiednikiem są wzorce uporządkowane
- Nie ma w niech slotów, a jedynie lista wartości
- Implementowane są jako wzorzec z jednym multislotem "__data"
- W tym multislocie przechowywane są wszystkie wartości

Definiowanie asercji

Jess

```
(deftemplate potencjalni
(declare (ordered TRUE)))
```

Java

```
Rete r = new Rete();
Deftemplate dt = new Deftemplate("potencjalni",
"Komentarz", r);
dt.addMultiSlot("__data", null, "ANY");
r.addDeftemplate(dt);
```

Fakty

- Fakty to instancje wzorców
- Wewnątrz systemu każdy fakt posiada liczbowy identyfikator
- Fakty mogą być zgrupowane w nazwane zbiory faktów
- Funkcja facts wypisuje wszystkie istniejące fakty
- Funkcja reset przywraca system do stanu początkowego, usuwając wszystkie tymczasowe fakty

Shadow facts

- Shadow facts to wskzanie na obiekty Java nie będące częścią systemu Jess, tj. nie znajdujące się w pamięci roboczej
- Umożliwiają one operacje na tych obiektach z poziomu systemu
- Nie jest konieczne ręczne definiowanie slotów; jeśli klasa jest zgodna z JavaBeans Jess stworzy je automatycznie dla każdej własności posiadającej metody get* i set*
- (deftemplate nazwa (declare (from-class nazwa-klasy))
- (defclass nazwa nazwa-klasy)
- W odróżnieniu od deftemplete defclass jest funkcją i może być użyta w dowolnym miejscu systemu
- defclass nie oferuje jednak wszystkich możliwości deftemplete
- Zalecane jest użycie defclass jeśli tylko jest to możliwe

Shadow facts — tworzenie i użycie

- (new nazwa-klasy) tworzy obiekt i zwraca referencję
- Referencję można traktować jak każdą inną wartość; najczęściej przypisuje się ją do zmiennej przy użyciu bind
- Aby dla obiektu do którego posiadamy referencję został stworzony shadow fact, musimy użyć funkcji add
- Funkcja ta doda nowy fakt do pamięci roboczej, dzięki czemu będzie można z niego skorzystać w regułach

Shadow facts — zmiany wartości

- Jeśli zmienimy shadow fact, zmiana ta zostanie przeniesiona również do obiektu
- Jeśli zmienimy obiekt bezpośrednio. shadow fact w pamięci roboczej nadal będzie przechowywał starą wartość
- Aby zaktualizować pojedynczy shadow fact należy użyć funkcji update
- Funkcja ta odczyta wartości z obiektu i przepisze je do faktu w pamięci roboczej
- Funkcja reset aktualizuje wszystkie shadow facts
- Jeśli chcemy aby Jess automatycznie aktualizował shadow facts, klasa którą mapujemy musi wspierać java.beans.PropertyChangeListener
- Dzięki temu Jess może dodać się do listy modułów powiadamianych o zmianie i będzie mógł automatycznie aktualizować pamięć roboczą

Tworzenie faktów

- Fakt może być stworzony przy użyciu funkcji assert; wówczas będzie faktem tymczasowym
- Fakty zgromadzone w zbiorach stworzonych przy użyciu funkcji deffacts są trwałe i nie są usuwane przez funkcją reset

Tworzenie faktów

Jess

```
(assert (towar-nietrwały (nazwa "raki")
(masa-jednostkowa 1.2)))
```

Java

```
Rete r = new Rete();
Fact f = new Fact("towar-nietrwały", r);
Value v0 = new Value("raki", RU.STRING);
f.setSlotValue("nazwa", v0);
Value v1 = new Value(1.2, RU.STRING);
f.setSlotValue("masa-jednostkowa", v1);
r.assertFact(f);
```

Tworzenie faktów

Jess

```
(deffacts system-przewoźnik
(faktura (skąd "Białystok") (dokąd "Mońki") (odległość 45)
(ekspress tak) (ilość 3) (towar "raki") (numer "11/99"))
(towar-nietrwały (nazwa "ryby") (masa-jednostkowa 2.9)))
```

Tworzenie asercji

Java

```
(assert (potencjalni ?f ?n ?cena))
```

Java

```
Rete r = new Rete();
Fact f = new Fact("potencjalni", r);
ValueVector vv = new ValueVector();
vv.add("?f").add("?n").add("?cena");
f.setSlotValue("__data", new Value(vv, RU.LIST));
r.assertFact(f);
```

Reguły

- Reguła składa się z dwóch części, oddzielonych =>
- Pierwsza część opisuje warunki jakie muszą być spełnione aby reguła została uruchomiona
- Zwykle są w niej opisane fakty z jakich korzysta reguła i warunki jakie muszą być spełnione przez te fakty
- Druga część opisuje działanie jakie musi zostać podjęte jeśli reguła jest ucuchamiana
- Może to być stworzenie, usunięcie lub zmiana faktu, wypisanie komunikatu, ...
- W drugiej części można również stworzyć nowe zmienne i/lub przypisać im wartości przy użyciu funkcji bind
- Funkcji tej nie można użyć w pierwszej części
- Reguła może mieć priorytet, opisany przez własność salience

Dopasowywanie faktów

- Dany fragment sprawdza typ faktu
- Reguła jest sprawdzana i wywoływana dla każdej kombinacji faktów zgodnych z przedstawionymi wzorcami
- Sprawdzany jest typ (wzorzec) faktu i dodatkowe warunki podane przez użytkownika
- Jeśli chcemy pobrać wskazanie na fakt, używamy operatort $<-$:
- ?f $<-$ (wzorzec ...)

Warunki dotyczące wartości slotów

- Aby warunek opisujący fakt był prawdziwy, muszą być spełnione wszystkie warunki dotyczące jego slotów
- Pobranie wartości slotu odbywa się poprzez napisanie w nawiasach nazwy slotu i zmiennej pod którą podstawiamy wartość: (slot ?zmienna)
- Trzy sposoby zapisu warunków
  1. Po pobraniu wszystkich faktów i ich wartości użycie funkcji test
  2. Wyrażenie otoczone nawiasami klamrowycmi umieszczone wewnątrz opisu faktu. W wyrażeniu tym można posługiwać się nazwami slotów i istniejących zmiennych. Nie tworzy ono żadnych nowych obiektów ani zmiennych
  3. Łącznie z pobieraniem wartości, umieszczając wyrażenie w notacji uproszczonej tuż po nazwie zmiennej
- Jeśli chcemy upewnić się, że wartość istnieje nie pobierając jej, zamiast “?zmienna” używamy tylko “?”.

Notacja uproszczona

~	negacja
&	koniunkcja
—	alternatywa
:(wyrażenie)	wykonanie wyrażenia zwracającego wartość logiczną
=(wyrażenie)	wykonanie wyrażenia zwracającego wartość
/wyrażenie/	użycie wyrażenia regularnego

Dopasowywanie multislotów

- Multislot przechowuje listę wartości
- Jest ona uporządkowana; należy o tym pamiętać podczas pobierania wartości i testów
- Jeśli chcemy pobrać nie pojedynczą wartość a listę, wyrażenie należy poprzedzić znakiem dolara "$"
- Dotyczy to również pojedynczego znaku zapytania; tu zamiast "?" piszemy "$?"
- Aby sprawdzić czy wartość należy do listy należy użyć funkcji (member$ wartość lista)

Operacje na faktach

- Aby móc wykonać operacje na faktach w części reguły po operatorze "=>" należy znać tożsamość faktu
- Pobieramy ją w pierwszej części reguły za pomocą operatora "− >"
- Nowy fakt tworzymy za pomocą funkcji assert, której jako parametr podajemy listę składającą się z nazwy ramy i listy par slot-wartość
- Fakty usuwane są za pomocą funkcji retract której jako parametr podajemy wskazania na fakt
- Aby zmienić wartości w slotach należy użyć funkcji modify, której pierwszym argumentem jest wskazanie na fakt, a jako kolejne argumenty występują pary slot-nowa wartość

Tworzenie reguł

Jess

```
(defrule sprawdź-wszystko
?w <- (wykonawcy (masa $? ?n $?) (zasięg $?zasięg)
(specjalne $?specjalne) (ekspress $?ekspress)
(wszystko $?slot))
(test (and (not (member$ ?n $?slot))
(member$ ?n $?zasięg)
(member$ ?n $?ekspress)
(member$ ?n $?specjalne)
))
=>
(modify ?w (wszystko $?slot ?n))
(printout t "Firma " ?n " jest w " $?slot crlf)
)
```

Tworzenie reguł

Java

Tworzenie reguł w języku Java nie jest zalecane.
Jest to nowa jeszcze nieudokumentowana funkcjonalność. Wraz z rozwojem systemu sposób tworzenia i obsługi reguł z poziomu języka Java może się zmianić.
Autorzy zaznaczają również, że istotą systemu ekspertowego jest oddzielenie reguł od kodu i danych, zaś użycie języka Java do tworzenia reguł może skomplikować system.

Wczytywanie pliku z kodem Jess

Java

```
Rete engine = new Rete();
FileReader file = new FileReader("myfile.clp");
try {
Jesp parser = new Jesp(file, engine);
parser.parse(false);
} finally {
file.close();
}
```

Wczytywanie pliku z kodem Jess

Java

```
Rete engine = new Rete();
FileReader file = new FileReader("myfile.clp");
Context context = engine.getGlobalContext();
try {
Jesp parser = new Jesp(file, engine);
Object result = Funcall.TRUE;
while (!result.equals(Funcall.EOF)) {
result = parser.parseExpression(context, false);
...
}
} finally {
file.close();
}
```

Uruchomienie systemu

- Aby uruchomić system, należy użyć funkcji run
- System będzie przeglądał reguły i wywoływał te, które mogą być wywołane
- Działanie skończy się gdy nie będzie możliwości wywołania żadnej reguły
- Przed run zwykle wywoływana jest funkcja reset, aby powrócić do stanu początkowego usuwając tymczasowe fakty
- Dzięki temu każde uruchomienie będzie rozpoczynało od tego samego stanu, co powinno zapewnić powtarzalność obserwacji

Baza wiedzy

- Zbiór faktów może być widziany jako baza danych
- Zatem możemy pobrać część obiektów
- Najprostsze wyjście to pobrać obiekty danego typu

Pobranie obiektów

Java

```
Rete engine = new Rete();
engine.batch("plik.clp");
Iterator it = engine.getObjects(new Filter() {
public boolean accept(Object o) {
return o instanceof Typ;
}
});
while (it.hasNext()) {
((Payment) it.next()).process();
}
```

Zapytania

- Aby pobrać obiekty spełniające określone warunki należy użyć zapytań
- Zapytania służą do zwracania wyników
- Reguły pozwalają na zmianę, zapytania tylko na odczyt
- Zapytanie może posiadać parametry

Stworzenie zapytania

Jess

```
(defquery znajdź-firmy-obsługujące (declare (variables ?m))
(firma (nazwa ?n) (zasięg ??m?) (ładowność ?l)))
```

Java

Podobnie jak w przypadku reguł, tworzenie zapytań w języku Java nie jest zalecane.

Użycie zapytań

Jess

```
(bind ?result (run-query* znajdź-firmy-obsługujące
"Białystok"))
(while (?result next)
(printout t (?result getString n) " " (?result getString n)
" " (?result getFloat l)))
```

Java

```
QueryResult result = engine.runQueryStar("znajdx-firmy",
new ValueVector().add("Białystok"));
while (result.next()) {
System.out.println(result.getString("n")
+ " " + result.getString("m")
+ " " + result.getFloat("l"));
}
```

Wnioskowanie wstecz

- Aby rama była użyta podczas wnioskowania wstecz, musi posiadać atrybut backchain-reactive ustawiony na true
- Ewentualnie, można włączyć po definicji przez użycie funkcji (do-backward-chaining wzorzec)
- Włączenie atrybutu wnioskowania wstecz zmienia strukturę reguł korzystających z takich ram i faktów
- W związku z tym należy zaznaczyć użycie ram we wnioskowania wstecz przed definicją reguł korzystających z nich korzystających

Reguły we wnioskowaniu wstecz

- Jeśli reguła usiłuje pobrać nieistniejący fakt którego wzorzec umożliwia wnioskowanie wstecz, tworzony jest nowy fakt
- Ten nowy fakt jest kopią niezbędnego faktu, gdzie zamiast nieznanych wartości umieszczana jest wartość nil, zaś jego nazwa powstaje poprzez dodanie na początku “need-”
- Aby zaimplementować wnioskowanie wstecz, należy stworzyć reguły które na podstawie faktów “need-*” stworzą niezbędne fakty
- W systemie będą zatem dwie klasy reguł — do wnioskowania w przód i do wnioskowania wstecz

Interfejs użytkownika

- Domyślnie system pracuje w trybie tekstowych, wypisując komunikaty na ekran i oczekując wprowadzenia wartości z klawiatury
- Istnieje możliwość implementacji interfejsu okienkowego
- Interfejs tworzony jest za pomocą biblioteki AWT, do której można odwołać się z poziomy języka Jess
- Przykład examples/jess/jframe.clp
- Stworzenie okna i przycisków
- Dodanie akcji wywoływanych po naciśnięciu przycisku
- Pokazanie okna

Tworzenie okna

Jess

```
(deffunction create-frame ()
(bind ?*f* (new JFrame "Jess Reflection Demo"))
(bind ?*c* (?*f* getContentPane))
(set ?*c* background (Color.magenta)))

(deffunction add-widgets ()
(?*c* add (new JLabel "This is: ") (BorderLayout.CENTER))
(bind ?*m* (new JComboBox))
(?*m* addItem "Cool")
(?*m* addItem "Really Cool")
(?*m* addItem "Awesome")
(?*c* add ?*m* (BorderLayout.SOUTH)))
```

Wyświeltanie, akcje

Jess

```
(deffunction add-behaviours ()
(?*f* setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE))
(?*m* addActionListener (implement ActionListener using
(lambda (?name ?event)
(printout t "You chose: " (get ?*m* selectedItem) crlf)))))

(deffunction show-frame ()
(?*f* pack)
(?*f* setVisible TRUE))

(create-frame)
(add-widgets)
(add-behaviours)
(show-frame)
```