Universidade do Minho

Exercício 2 – Extensão à Programação em Lógica e Conhecimento Imperfeito

Mestrado Integrado em Engenharia Informática - MiEI

Sistemas de Representação de Conhecimento e Raciocínio
(2º Semestre/2015-2016)

A72223 Gustavo da Costa Gomes
A71223 José Carlos da Silva Brandão Gonçalves
A70443 Tiago João Lopes Carvalhais

Local - Braga
Data - 25/04/2016

Resumo

Este documento serve de apoio ao segundo exercício prático da unidade curricular de Sistemas de Representação de Conhecimento e Raciocínio, cujo objetivo principal é o de motivar para a utilização da linguagem de programação em lógica estendida, recorrendo ao *PROLOG* e com as devidas alterações implementadas.

Numa primeira fase, serão explicadas as estratégias definidas para representar a informação pretendida, constituindo assim a base de conhecimento para este trabalho. De seguida, serão abordados todos os objetivos propostos no enunciado, bem como as técnicas utilizadas para os solucionar. Foram também desenvolvidas algumas *queries*, a fim de enriquecer este exercício e, para ajudar, foram utilizados alguns predicados auxiliares, pelo que também será efetuada uma explicação pormenorizada acerca dos mesmos. Para além disso, foi-nos proposta também a interação com o sistema, utilizando para tal, uma janela desenvolvida em *JAVA SWING*, que iremos também descrever neste relatório.

Num fase mais terminal deste relatório, serão ainda colocados exemplos de como questionar a base de conhecimento acerca das funcionalidades criadas, bem como os resultados produzidos pela mesma na janela desenvolvida.

Índice

1. Introdução

O objetivo deste trabalho é a utilização da linguagem de programação em lógica estendida, *PROLOG* , com as devidas modificações por forma a conseguir representar conhecimento perfeito e imperfeito e criar mecanismos que atuem sobre o mesmo, efetuando uma interação com outra linguagem e paradigma de programação.

O conhecimento perfeito caracteriza-se por representar factos certos, ou seja, não deixa dúvidas sobre a veracidade de uma afirmação. Já o conhecimento imperfeito deixa algumas incertezas, isto é, não é seguro que uma dada afirmação seja verdadeira ou falsa, pelo que se introduz um terceiro valor de verdade, que se junta ao verdadeiro e ao falso: o desconhecido. Este tipo de conhecimento possui três variantes, que passamos a explicar de seguida.

O conhecimento imperfeito do tipo incerto ocorre quando não se sabe determinado facto, mas esse pode vir a ser descoberto, sendo que, enquanto não o for, qualquer resposta pode ser verdadeira ou falsa, portanto, será representada como desconhecida. Por exemplo, o Tiago recebeu de salário um valor que ainda ninguém conhece, com exceção do próprio, pelo que poderá revelar a quantia que recebeu assim que o desejar, sendo que, até lá, esse valor permanece desconhecido.

O conhecimento imperfeito do tipo impreciso acontece numa situação em que a dúvida incide num determinado intervalo de incerteza. Por exemplo, não se sabe quanto dinheiro recebeu o José, mas sabe-se que foi um valor superior a 500€ e inferior a 600€. Qualquer valor que esteja incluído dentro deste intervalo será representado como desconhecido e qualquer outro valor será falso.

Por fim, o conhecimento imperfeito do tipo interdito é similar à primeira variante, contudo, neste tipo de situações, é impossível vir-se a esclarecer a dúvida existente, pelo que esta permanecerá permanentemente uma incógnita. A título de exemplo, o Gustavo transportava uma carteira com uma elevada quantidade monetária, que nem o próprio sabia ao

certo o seu valor, contudo, esta cai dentro de uma sarjeta e o dinheiro desaparece, sendo que, se ninguém sabia quanto havia na mesma, esse valor será impossível de saber.

Ao longo deste trabalho prático, irão surgir vários exemplos destes tipos de conhecimento exemplificados acima, adaptados à temática sugerida pela equipa docente da unidade curricular.

2. Preliminares

Posto isto, passemos, então, a um breve resumo sobre a temática do segundo exercício do trabalho prático. Mais uma vez e, à semelhança do que foi o primeiro exercício, o tema incide no registo de eventos em instituições de saúde. Contrariamente à fase anterior, em que foi o grupo a decidir que predicados apresentar e como os representar, foi a equipa docente a definir os mesmos e a maneira como são representados. Sendo assim, iremos desde já passar a uma breve explicação sobre estes.

Um utente é composto por um identificador, um nome, uma idade e uma morada, sendo que o primeiro deverá ser único, ou seja, não poderá existir mais nenhum utente com o mesmo número, podendo contudo possuir o mesmo nome, a mesma idade e até a mesma morada.

Um serviço é constituído por um identificador, uma designação, uma instituição e uma cidade. À semelhança do utente, também um serviço deve possuir um identificador único, podendo os outros parâmetros ser iguais.

Por fim, a consulta possui uma data, um número do utente que a efetuou, um identificador do respetivo serviço em que se insere e um custo. Contrariamente aos dois predicados anteriores, uma consulta, como não possui um identificador próprio, a sua unicidade terá em conta os atributos da data, número de utente e número de serviço, pelo que, nenhuma consulta pode ter a mesma data, utente e serviço ao mesmo tempo, sendo o custo irrelevante neste caso.

O grupo decidiu incluir no código do trabalho prático, dois exemplos de conhecimento perfeito, conhecimento imperfeito incerto, impreciso e interdito e um exemplo de conhecimento negativo, para cada um dos predicados. Para se efetuarem questões ao sistema, a partir de agora, deve-se utilizar a função `demo`, que será explicada mais à frente.

3. Código *PROLOG*

3.0. Grau de confiança das principais extensões em lógica

3.0.1 Predicado “Consulta”

Grau de Confiança:

{

-consulta ((QoI$_D$, DoC$_D$), (QoI$_{IdU}$, DoC$_{IdU}$), (QoI$_{Ids}$, DoC$_{Ids}$), (QoI$_C$, DoC$_C$))
← não consulta((QoI$_D$, DoC$_D$), (QoI$_{IdU}$, DoC$_{IdU}$), (QoI$_{Ids}$, DoC$_{Ids}$), (QoI$_C$, DoC$_C$)))

exception$_{consulta}$((QoI$_\perp$, DoC$_\perp$), (QoI$_{IdU}$, DoC$_{IdU}$), (QoI$_{IdS}$, DoC$_{IdS}$), (QoI$_C$, DoC$_C$)) :: QoI :: DoC

exception$_{consulta}$((QoI$_D$, DoC$_D$), (QoI$_{IdU}$, DoC$_{IdU}$), (QoI$_{IdS}$, DoC$_{IdS}$), (QoI$_\perp$, DoC$_\perp$)) :: QoI :: DoC

exception$_{consulta}$((QoI$_{\{20\text{-}7\text{-}2015\}}$, DoC$_{\{20\text{-}7\text{-}2015\}}$), (QoI$_5$, DoC$_5$), (QoI$_5$, DoC$_5$), (QoI$_{25}$, DoC$_{25}$)) :: QoI :: DoC

exception$_{consulta}$((QoI$_{\{21\text{-}7\text{-}2015\}}$, DoC$_{\{21\text{-}7\text{-}2015\}}$), (QoI$_5$, DoC$_5$), (QoI$_5$, DoC$_5$), (QoI$_{25}$, DoC$_{25}$)) :: QoI :: DoC

exception$_{consulta}$((QoI$_{\{9\text{-}9\text{-}2013\}}$, DoC$_{\{9\text{-}9\text{-}2013\}}$), (QoI$_6$, DoC$_6$), (QoI$_2$, DoC$_2$), (QoI$_{30}$, DoC$_{30}$)) :: QoI :: DoC

exception$_{consulta}$((QoI$_{\{9\text{-}9\text{-}2013\}}$, DoC$_{\{9\text{-}9\text{-}2013\}}$), (QoI$_6$, DoC$_6$), (QoI$_2$, DoC$_2$), (QoI$_{20}$, DoC$_{20}$)) :: QoI :: DoC

exception$_{consulta}$((QoI$_{\{7\text{-}12\text{-}2015\}}$, DoC$_{\{7\text{-}12\text{-}2015\}}$), (QoI$_4$, DoC$_4$), (QoI$_3$, DoC$_3$), (QoI$_\perp$, DoC$_\perp$)) :: QoI :: DoC

exception$_{consulta}$((QoI$_\perp$, DoC$_\perp$), (QoI$_1$, DoC$_1$), (QoI$_4$, DoC$_4$), (QoI$_{10}$, DoC$_{10}$)) :: QoI :: DoC

}:: 1

3.0.2 Predicado “Utente”

Grau de Confiança:

{

-utente ((QoI$_{IdU}$, DoC$_{IdU}$), (QoI$_{N}$, DoC$_{N}$), (QoI$_{I}$, DoC$_{I}$), (QoI$_{M}$, DoC$_{M}$))
← não utente((QoI$_{IdU}$, DoC$_{IdU}$), (QoI$_{N}$, DoC$_{N}$), (QoI$_{I}$, DoC$_{I}$), (QoI$_{M}$, DoC$_{M}$))

exception$_{utente}$((QoI$_{IdU}$, DoC$_{IdU}$), (QoI$_{N}$, DoC$_{N}$), (QoI$_{I}$, DoC$_{I}$), (QoI$_{M}$, DoC$_{M}$)) :: QoI :: DoC

exception$_{utente}$((QoI$_{5}$, DoC$_{5}$), (QoI$_{\{mario_cardoso\}}$, DoC$_{\{mario_cardoso\}}$), (QoI$_{34}$, DoC$_{34}$), (QoI$_{\{rua_das_tristezas\}}$, DoC$_{\{rua_das_tristezas\}}$)) :: QoI :: DoC
exception$_{utente}$((QoI$_{5}$, DoC$_{5}$), (QoI$_{\{mario_cardoso\}}$, DoC$_{\{mario_cardoso\}}$), (QoI$_{35}$, DoC$_{35}$), (QoI$_{\{rua_das_tristezas\}}$, DoC$_{\{rua_das_tristezas\}}$)) :: QoI :: DoC

exception$_{utente}$((QoI$_{6}$, DoC$_{6}$), (QoI$_{\{joel_vaz\}}$, DoC$_{\{joel_vaz\}}$), (QoI$_{56}$, DoC$_{56}$), (QoI$_{\{rua_das_estrelas\}}$, DoC$_{\{rua_das_estrelas\}}$)) :: QoI :: DoC
exception$_{utente}$((QoI$_{6}$, DoC$_{6}$), (QoI$_{\{joel_vaz\}}$, DoC$_{\{joel_vaz\}}$), (QoI$_{56}$, DoC$_{56}$), (QoI$_{\{rua_das_luas\}}$, DoC$_{\{rua_das_luas\}}$)) :: QoI :: DoC

exception$_{utente}$((QoI$_{Id}$, DoC$_{Id}$), (QoI$_{N}$, DoC$_{N}$), (QoI$_{\perp}$, DoC$_{\perp}$), (QoI$_{M}$, DoC$_{M}$)) :: QoI :: DoC
exception$_{utente}$((QoI$_{Id}$, DoC$_{Id}$), (QoI$_{N}$, DoC$_{N}$), (QoI$_{I}$, DoC$_{I}$), (QoI$_{\perp}$, DoC$_{\perp}$)) :: QoI :: DoC

exception$_{utente}$((QoI$_{7}$, DoC$_{7}$), (QoI$_{\{jose_esteves\}}$, DoC$_{\{jose_esteves\}}$), (QoI$_{I}$, DoC$_{I}$), (QoI$_{\{rua_das_alcatifas\}}$, DoC$_{\{rua_das_alcatifas\}}$)) :: QoI :: DoC
exception$_{utente}$((QoI$_{8}$, DoC$_{8}$), (QoI$_{\{otavio_correia\}}$, DoC$_{\{otavio_correia\}}$), (QoI$_{79}$, DoC$_{79}$), (QoI$_{M}$, DoC$_{M}$)) :: QoI :: DoC

}:: 1

3.0.3 Predicado “Servico”

Grau de Confiança:

{

-servico ((QoI$_{IdS,}$ DoC$_{IdS}$), (QoI$_{D}$, DoC$_{D}$), (QoI$_{I}$, DoC$_{I}$), (QoI$_{C}$, DoC$_{C}$))
← não servico((QoI$_{IdS,}$ DoC$_{IdS}$), (QoI$_{D}$, DoC$_{D}$), (QoI$_{I}$, DoC$_{I}$), (QoI$_{C}$, DoC$_{C}$))

exception$_{servico}$((QoI$_{IdS,}$ DoC$_{IdS}$), (QoI$_{D}$, DoC$_{D}$), (QoI$_{\perp}$, DoC$_{\perp}$), (QoI$_{C}$, DoC$_{C}$)) :: QoI :: DoC
exception$_{servico}$((QoI$_{IdS,}$ DoC$_{IdS}$), (QoI$_{D}$, DoC$_{D}$), (QoI$_{I}$, DoC$_{I}$), (QoI$_{\perp}$, DoC$_{\perp}$)) :: QoI :: DoC

exception$_{servico}$((QoI$_{5,}$ DoC$_{5}$), (QoI$_{\{cardiologia\}}$, DoC$_{\{cardiologia\}}$), (QoI$_{hospital_faro}$, DoC$_{hospital_faro}$), (QoI$_{faro}$, DoC$_{faro}$)) :: QoI :: DoC
exception$_{servico}$((QoI$_{5,}$ DoC$_{5}$), (QoI$_{\{urologia\}}$, DoC$_{\{urologia\}}$), (QoI$_{hospital_faro}$, DoC$_{hospital_faro}$), (QoI$_{faro}$, DoC$_{faro}$)) :: QoI :: DoC

exception$_{servico}$((QoI$_{6,}$ DoC$_{6}$), (QoI$_{\{psiquiatria\}}$, DoC$_{\{psiquiatria\}}$), (QoI$_{hospital_militar}$, DoC$_{hospital_militar}$), (QoI$_{lisboa}$, DoC$_{lisboa}$)) :: QoI :: DoC
exception$_{servico}$((QoI$_{6,}$ DoC$_{6}$), (QoI$_{\{psiquiatria\}}$, DoC$_{\{psiquiatria\}}$), (QoI$_{hospital_militar}$, DoC$_{hospital_militar}$), (QoI$_{porto}$, DoC$_{porto}$)) :: QoI :: DoC

exception$_{servico}$((QoI$_{7,}$ DoC$_{7}$), (QoI$_{\{psicologia\}}$, DoC$_{\{psicologia\}}$), (QoI$_{\{hospital_amadora_sintra\}}$, DoC$_{\{hospital_amadora_sintra\}}$), (QoI$_{C}$, DoC$_{C}$)) :: QoI :: DoC
exception$_{servico}$((QoI$_{IdS,}$ DoC$_{IdS}$), (QoI$_{\{oftalmologia\}}$, DoC$_{\{oftalmologia\}}$), (QoI$_{\{hospital_sao_joao\}}$, DoC$_{\{hospital_sao_joao\}}$), (QoI$_{porto}$, DoC$_{porto}$)) :: QoI :: DoC

} :: 1

3.0.4 Predicado “demo”

```
demo <- (Condição -> Ação) ,
         Verificar(Condição),
                 Executar(Ação).

Verificar(Condição)
      -Verificar(X) <- nao(Verificar(X)).
      Verificar([]).
      Verificar([X|T]) <- X , Verificar(T).

Executar([]) <- demo.
Executar([STOP]).
Executar([X|Y]) <- X ,  Executar(Y).
```

O predicado demo funciona da maneira descrita acima, onde para cada par condição-ação primeiramente verifica-se se não viola nenhum invariante, e nesse caso proceder-se-á então á execução da ação associada a essa condição.

3.1. Base de Conhecimento e Representação

Para a representação dos vários tipos de conhecimento existentes neste trabalho prático, foram criados na base de conhecimento factos para utentes, serviços e consultas. Como já foi dito na secção anterior, um utente é caracterizado pelo seu identificador, nome, idade e morada, estando escrito da seguinte forma:

```
utente(1, manuel_faria, 24, rua_das_papoilas).
```

Ao observarmos este facto, podemos referir que o utente número 1 tem o nome de `manuel_faria`, tem 24 anos e reside na `rua_das_papoilas`.

Já um serviço requer um identificador, designação, instituição e cidade, sendo representado da seguinte forma:

```
servico(1, oncologia, hospital_braga, braga).
```

Desta forma, podemos afirmar que o serviço número 1 é de `oncologia` e existe na instituição `hospital_braga`, na cidade de `braga`.

Por fim, uma consulta possui uma data, número de utente, identificador de serviço e custo, sendo representada desta forma:

```
consulta(18-4-2015, 1, 1, 25).
```

Podemos, então, referir que esta consulta foi efetuada na data de `18-4-2015`, pelo utente número 1, no serviço 1 e custou 25 unidades monetárias.

Contudo, deparamo-nos agora com um problema: como representar conhecimento desconhecido? Deixamos, neste momento, de estar no pressuposto do mundo fechado, onde o conhecimento ou é verdadeiro ou falso, e entramos num pressuposto de mundo aberto, com a introdução de um terceiro valor de verdade: o desconhecido. Iremos agora necessitar de colocar questões ao sistema, de modo a que o conjunto de respostas englobe estes três valores de verdade, pelo que tivemos de definir uma nova função: a *demo*.

```
demo(Q, verdadeiro) :- Q.
demo(Q, falso) :- -Q.
demo(Q, desconhecido) :- nao(Q), nao(-Q).
```

Como podemos verificar, se uma questão tem prova na base de conhecimento, então o valor de verdade será verdadeiro. Caso exista uma prova de que essa questão é falsa, então o valor de verdade retornado será falso. Caso não exista nem uma prova positiva, nem uma prova negativa dessa questão, então o resultado será desconhecido.

Também podemos querer saber se um conjunto de questões são verdadeiras, falsas ou desconhecidas, pelo que optamos pela criação de uma função da família da *demo*, denominada *demoExtendido*.

```
demoExtendido(Q1 e Q2, R) :- demo(Q1, R1), demoExtendido(Q2, R2),
                             e(R1, R2 ,R).
demoExtendido(Q1, R1) :- demo(Q1, R1).
```

O predicado `e`, explicado em anexo, funciona como uma espécie de tabela de verdade que irá verificar se, entre um conjunto de duas questões, o resultado é verdadeiro, falso ou desconhecido. O programa recebe uma questão `Q1` e um conjunto de questões `Q2` e irá começar por chamar a função *demo* para calcular o valor de verdade da primeira questão e guarda o resultado em `R1`, sendo que, seguidamente será chamada a própria função de forma recursiva para calcular o valor de verdade do conjunto de questões e no fim juntam-se os dois resultados e determina-se o valor lógico resultante. E o operador ‘e’ é o símbolo escolhido para separar as questões que são recebidas como um só argumento e foi definida da mesma forma que o operador ‘::’ utilizado pelos invariantes.

Para além disso, devemos também referir quando um predicado deve devolver falso. Essa situação deve ocorrer quando não existe prova de que a questão é verdadeira e de

quando não exista uma exceção que indique que um certo valor dessa questão seja nulo, para evitar que em situações em que o resultado deva ser desconhecido, não devolva falso.

```
-utente(IdU, N, I, M) :- nao(utente(IdU, N, I, M)),
                         nao(exception(utente(IdU, N, I, M))).

-servico(IdS, D, I, C) :- nao(servico(IdS, D, I, C)),
                          nao(exception(servico(IdS, D, I, C))).

-consulta(D, IdU, IdS, C) :- nao(consulta(D, IdU, IdS, C)),
                             nao(exception(consulta(D, IdU, IdS, C))).
```

3.2. Exemplos de Conhecimento Perfeito

Como referirmos anteriormente, o conhecimento perfeito caracteriza-se por ter um grau de certeza de 100%, pelo que é algo que podemos afirmar como sendo verdadeiro. Seguem em baixo todos os predicados que foram inseridos na base de conhecimento e que se enquadram neste tipo de conhecimento.

```
utente(1, manuel_faria, 24, rua_das_papoilas).
utente(2, carlos_sousa, 45, rua_dos_malmequeres).

servico(1, oncologia, hospital_braga, braga).
servico(2, pediatria, hospital_porto, porto).

consulta(18-4-2015, 1, 1, 25).
consulta(25-2-2016, 2, 3, 30).
```

3.3. Exemplos de Conhecimento Negativo

Nos exemplos anteriores demonstramos vários casos de factos positivos, ou seja, conhecimento que podemos afirmar que é verdadeiro, mas também podemos inserir factos negativos, ou seja, dizer claramente que algo é mentira. Para tal, utilizámos um carater de negação no início dos predicados, como está explícito em baixo.

```
-utente(5, mario_cardoso, 33, rua_das_tristezas).

-servico(6, psiquiatria, hospital_militar, braga).

-consulta(22-7-2015, 5, 5, 25).
```

3.4. Exemplos de Conhecimento Imperfeito Incerto

Numa outra perspectiva temos o conhecimento imperfeito, que é caracterizado pela falta de certeza acerca de um predicado, sendo que o do tipo incerto tem a particularidade de possuir valores desconhecidos, definidos previamente como sendo valores nulos. Exemplos deste tipo estão presentes em baixo.

```
utente(3, joao_seabra, xpto1, rua_da_alegria).
utente(4, tiago_barbosa, 37, xpto2).

exception(utente(Id, N, I, M)) :- utente(Id, N, xpto1, M).
exception(utente(Id, N, I, M)) :- utente(Id, N, I, xpto2).

nulo(xpto1).
nulo(xpto2).
```

Como se pode verificar, no caso do primeiro utente, a idade é desconhecida, sendo esta definida como um valor nulo, de nome `xpto1`. Como queremos que a função *demo* caracterize este predicado como sendo desconhecido, devemos acrescentar uma exceção que diga que, se for encontrado esse valor nulo, diga que este predicado apresenta como valor de verdade o desconhecido. Seguem em baixo mais exemplos para os restantes predicados, usando o mesmo raciocínio.

```
servico(3, cirurgia, xpto3, lisboa).
servico(4, radiologia, ipo_porto, xpto4).

exception(servico(IdS, D, I, C)) :- servico(IdS, D, xpto3, C).
exception(servico(IdS, D, I, C)) :- servico(IdS, D, I, xpto4).

nulo(xpto3).
nulo(xpto4).
```

```
consulta(xpto5, 3, 1, 25).
consulta(30-5-2014, 4, 2, xpto6).


exception(consulta(D, IdU, IdS, C)) :- consulta(xpto5, IdU, IdS, C).
exception(consulta(D, IdU, IdS, C)) :- consulta(D, IdU, IdS, xpto6).


nulo(xpto5).
nulo(xpto6).
```

3.5. Exemplos de Conhecimento Imperfeito Impreciso

O conhecimento imperfeito do tipo impreciso é caracterizado pela sua resposta estar contida dentro de um determinado intervalo de incerteza, sendo que qualquer valor dentro do mesmo será representado como desconhecido e qualquer outro como falso. Abaixo seguem alguns exemplos.

```
exception(servico(5, cardiologia, hospital_faro, faro)).
exception(servico(5, urologia, hospital_faro, faro)).
```

Á semelhança do caso anterior, foram criadas exceções para referir que o serviço número 5, existente na instituição `hospital_faro` na cidade de `faro`. O problema é que não se sabe ao certo qual a designação do serviço, mas sabe-se que, ou é de `cardiologia` ou de `urologia`. Ao colocar como exceção, o sistema irá reconhecer esta questão como desconhecida, desde que as designações sejam aquelas que foram referidas, caso contrário irá devolver como resposta o valor falso.

```
-servico(6, psiquiatria, hospital_militar, braga).
exception(servico(6, psiquiatria, hospital_militar, lisboa)).
exception(servico(6, psiquiatria, hospital_militar, porto)).
```

No exemplo acima, verificamos que o serviço número 6 é de `psiquiatria` e é efetuado na instituição `hospital_militar`, contudo não se sabe exatamente a cidade onde se situa a mesma, mas sabe-se que não é em `braga`, podendo estar situada em `lisboa` ou no `porto`.

Seguem em baixo exemplos para os restantes casos.

```
-utente(5, mario_cardoso, 33, rua_das_tristezas).
exception(utente(5, mario_cardoso, 34, rua_das_tristezas)).
exception(utente(5, mario_cardoso, 35, rua_das_tristezas)).

exception(utente(6, joel_vaz, 56, rua_das_estrelas)).
exception(utente(6, joel_vaz, 56, rua_das_luas)).


-consulta(22-7-2015, 5, 5, 25).
exception(consulta(20-7-2015, 5, 5, 25)).
exception(consulta(21-7-2015, 5, 5, 25)).

exception(consulta(9-9-2013, 6, 2, 30)).
exception(consulta(9-9-2013, 6, 2, 20)).
```

3.6. Exemplos de Conhecimento Imperfeito Interdito

Este tipo de conhecimento caracteriza-se de forma semelhante ao incerto, pois há um certo atributo que é desconhecido, contudo este nunca será conhecido, pelo que é necessário que o sistema garanta que não será possível a inserção de um valor para esse atributo, como está exemplificado em baixo.

```
exception(consulta(7-12-2015, 4, 3, C)).
exception(consulta(D, 1, 4, 10)).


+consulta(7-12-2015, 4, 3, C) :: (findall((7-12-2015, 4, 3, C),
                                  consulta(7-12-2015, 4, 3, C), S),
                                  length(S, T), T == 0).
+consulta(D, 1, 4, 10) :: (findall((D, 1, 4, 10),
                           consulta(D, 1, 4, 10), S),
                           length(S, T), T == 0).
```

Como podemos verificar, foram criadas mais exceções, para garantir que o predicado inserido seja reconhecido como desconhecido. Contudo, para garantir que a informação desconhecida não seja atualizada com inserção de conhecimento, foram criados invariantes

para garantir que, ao inserir o mesmo predicado, mas com um valor certo, neste caso, para o custo da consulta, este não seja inserido na base de conhecimento e se mantenha a incerteza.

Em baixo seguem os restantes exemplos.

```
exception(utente(7, jose_esteves, I, rua_das_alcatifas)).
exception(utente(8, otavio_correia, 79, M)).

+utente(7, jose_esteves, I, rua_das_alcatifas) :: (findall((7,
jose_esteves, I, rua_das_alcatifas), utente(7, jose_esteves, I,
rua_das_alcatifas), R), length(R, T), T == 0).

+utente(8, otavio_correia, 79, M) :: (findall((8, otavio_correia, 79,
M), utente(8, otavio_correia, 79, M), R), length(R, T), T == 0).

exception(servico(7, psicologia, hospital_amadora_sintra, C)).
exception(servico(IdS, oftalmologia, hospital_sao_joao, porto)).

+servico(7, psicologia, hospital_amadora_sintra, C) :: (findall((7,
psicologia, hospital_amadora_sintra, C), servico(7, psicologia,
hospital_amadora_sintra, C), S), length(S, T), T == 0).

+servico(IdS, oftalmologia, hospital_sao_joao, porto) ::
(findall((IdS, oftalmologia, hospital_sao_joao, porto), servico(IdS,
oftalmologia, hospital_sao_joao, porto), S), length(S, T), T == 0).
```

3.7. Resolução de *Queries*

Procedemos também à realização de três *queries*, um pouco à semelhança do primeiro exercício, a fim de enriquecer um pouco o trabalho realizado e para o utilizador da interface poder interagir com o sistema para poder consultar alguns dados estatísticos. As três funções estão representadas em baixo.

```
mediaIdades(R) :- findall(I, utente(IdU, N, I, M), S), media(S, R).
```

A função acima calcula a média de idades de todos os utentes. Começa por encontrar uma todos os que estão registados na base de conhecimento e, de seguida, constrói uma lista com as idades dos mesmos e calcula a média desses valores. Convém notar que as idades só

irão ser contabilizadas na contagem, apenas no caso de serem valores não nulos, ou seja, valores que sejam conhecidos.

```
servicosInstituicao(I, R) :- findall((IdS, D),
                             servico(IdS, D, I, C), R).
```

Esta função determina a lista de serviços que são efetuados numa dada instituição, sendo que o programa irá construir uma lista com todos os serviços registados e devolvê-la como *output*.

```
getConsultas(IdU, Lc, Tp) :- findall((D, C),
                             consulta(D, IdU, IdS, C), S),
                             filtraConsultas(S, Lc, Tp).
```

A função acima tem como função devolver uma lista com as consultas de um utilizador e o custo total das mesmas. Começa por determinar uma lista com as consultas registadas com o número do utilizador e depois chama a função `filtraConsultas`, que irá filtrar as consultas do utente que não possuem um valor de custo ou data desconhecidos.

3.8. Inserção e Remoção de Conhecimento

Como restrições à inserção e remoção de informação na base de conhecimento, decidimos proceder à criação de vários invariantes que nos garantam que as seguintes condições são verificadas:

- Não existir mais do que um utente com o mesmo identificador;
- Não existir mais do que um serviço com o mesmo identificador;
- Não existir mais do que uma consulta com a mesma data, número de cliente e número de serviço;
- Ao inserir uma consulta com um número de utente e serviço, estes já devem estar representados na base de conhecimento;
- Para se remover um serviço, este não deverá possuir consultas associadas;
- Para se remover um utente, este não deverá possuir consultas associadas;
- Não é permitido adicionar exceções repetidas.

Mediante estas condições, procedemos, então, à criação dos seguintes invariantes:

```
+utente(IdU, N, I, M) :: (findall((IdU, N2, I2, M2),
```

```
                          utente(IdU, N2, I2, M2), S),
                          length(S, T), T == 1).

+servico(IdS, D, I, C) :: (findall((IdS, D2, I2, C2),
                           servico(IdS, D2, I2, C2), S),
                           length(S, T), T == 1).

+consulta(D, IdU, IdS, C) :: (findall((D, IdU, IdS, C2),
                              consulta(D, IdU, IdS, C2), S),
                              length(S, T), T == 1).

+consulta(D, IdU, IdS, C) :: (findall(IdU, utente(IdU, N, I, M), S),
                              length(S, T), T == 1).

+consulta(D, IdU, IdS, C) :: (findall(IdS,
                              servico(IdS, Descr, I, Cid), S),
                              length(S, T), T == 1).

-servico(IdS, D, I, C) :: (findall(IdS,
                           consulta(Data, IdU, IdS, Custo), S),
                           length(S, T), T == 0).

-utente(IdU, N, I, M) :: (findall(IdU,
                          consulta(Data, IdU, IdS, Custo), S),
                          length(S, T), T == 0).

+exception(Q) :: (findall(Q, exception(Q), S), length(S, T), T == 1).
```

4. Interface Gráfica

A interface gráfica é composta por uma janela que dá a possibilidade aos utilizadores realizarem as suas queries de uma forma mais amigável em relação á interação utilizador-sistema fornecida pelo *SICStus* durante a realização das mesmas e foi utilizada a biblioteca *JASPER* como suporte ás interações entre o *SICStus* e o *JAVA*.

O utilizador primeiramente terá de escolher um ficheiro com a extensão .pl que contenha todo o conjunto de predicados dos quais pretenda efetuar questões, caso o utilizador não pretenda escrever o nome do ficheiro tem a possibilidade de carregar num botão 'Procurar Ficheiro' que lhe abre uma nova janela com os ficheiros todos disponíveis na diretoria atual e onde o utilizador poderá então selecionar o ficheiro pretendido. Existe também um ficheiro que é lido por defeito, isto é, no caso de um utilizador não inserir o nome nem de procurar o mesmo é lido o ficheiro que contém todo o conjunto de predicados já explicados na secção anterior do presente relatório. Após esta fase inicial o utilizador terá de clicar no botão 'Leitura' que dá inicio ao processamento do ficheiro, ou seja, torna o ficheiro disponível para a realização de queries. É possível ver estas situações nas figuras 1 e 2.

A partir deste momento o utilizador está pronto para conseguir efetuar qualquer query para a qual pretenda saber a resposta. E tem três maneiras de o fazer consoante o predicado em questão, e a escolha do tipo de query pode ser observado na figura 3.

Isto é, no caso, do predicado ser um predicado que calcula o valor de verdade de um ou mais atributos, por exemplo, o predicado, utente(1, manuel_faria, 24, rua_das_papoilas). que por sua vez dará como resultado ou 'verdadeiro' ou 'falso' ou ainda 'desconhecido', de acordo com o sistema de inferência novo, o utilizador terá de escolher a opção 'Valor de Verdade' como sendo o tipo de query. E após essa escolha será disponibilizado o conjunto de predicados pré-disponíveis cujo utilizador apenas terá de preencher os respetivos argumentos/atributos nos campos que serão indicados. Mas também poderá fazer esta na query caso escolha a opção 'Outra' como sendo o tipo de query. Esta opção permite ao utilizador escrever qualquer query completa tal e qual como o faria na linha de comandos disponibilizada pelo SICStus, e obter a resposta pretendida, mas nesta opção caso seja uma query do tipo valor de verdade será necessário utilizar a query como argumento do predicado demo, que constitui o novo sistema de inferência. Isto é, a query anterior teria de ser colocada desta forma, demo(utente(1, manuel_faria, 24, rua_das_papoilas),R). no campo indicado. E para qualquer outra query o mesmo não se aplica, por exemplo, a query utente(A,B,C,D). pode ser inserida dessa forma porque esta irá devolver os valores todos que existem na base de conhecimento sobre o utente e não um valor de verdade que indica se um determinado utente existe ou não na mesma e pode ser visto na figura 5.

No caso de ser um predicado normal, ou seja, um predicado que devolve todos os valores existentes na base de conhecimento o utilizador terá de escolher a opção 'Normal' como sendo o tipo de query e após essa escolha será disponibilizado a lista de predicados que poderão ser usados. E o utilizador apenas terá de escolher o predicado pretendido da lista

disponibilizada e de seguida inserir nos campos indicados os atributos dessa query e obterá a resposta. No caso de o predicado não existir na lista, o utilizador deverá escolher a opção 'Outra' e da mesma forma já explicada executar a query, e este tipo de query pode ser vista na figura 4.

E ainda tem a hipótese de escolher a opção 'Outra' que simplifica bastante a interface dando a possibilidade ao utilizador de apenas inserir a query completa tal e qual o faria na linha de comandos do *SICStus* e obter de imediato a sua resposta. Mas o utilizador precisa de ter cuidado ao inserir o seu predicado no caso de ser um predicado cuja resposta seja um valor de verdade o utilizador terá de efetuar essa query como sendo um argumento do predicado demo que por sua vez constitui o novo sistema de inferência, o exemplo do utente dado anteriormente demonstra exatamente o procedimento aqui explicado. Esta query pode ser observada na figura 6.

Os predicados disponibilizados de acordo com as opções 'Valor de Verdade' e 'Normal' apenas são os predicados principais e podem ser vistos nas imagens a seguir.

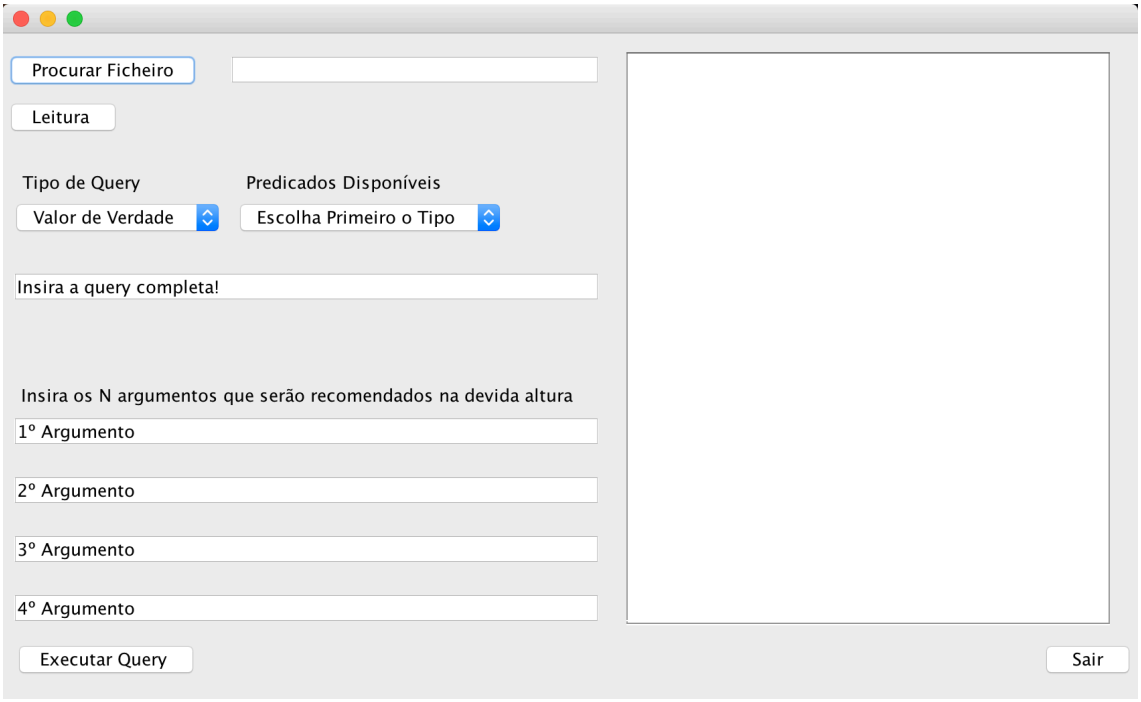

Figure 1 - Inicio

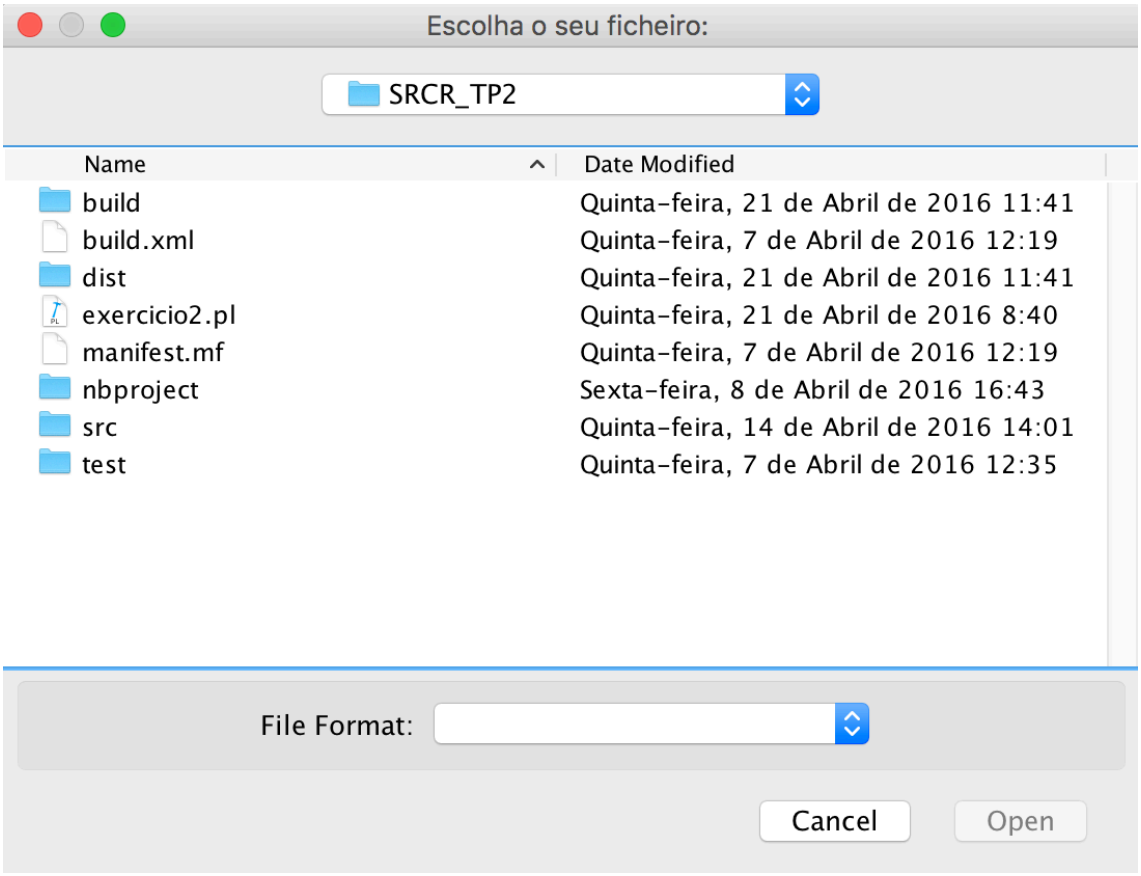

Figure 2-Procura de um Ficheiro

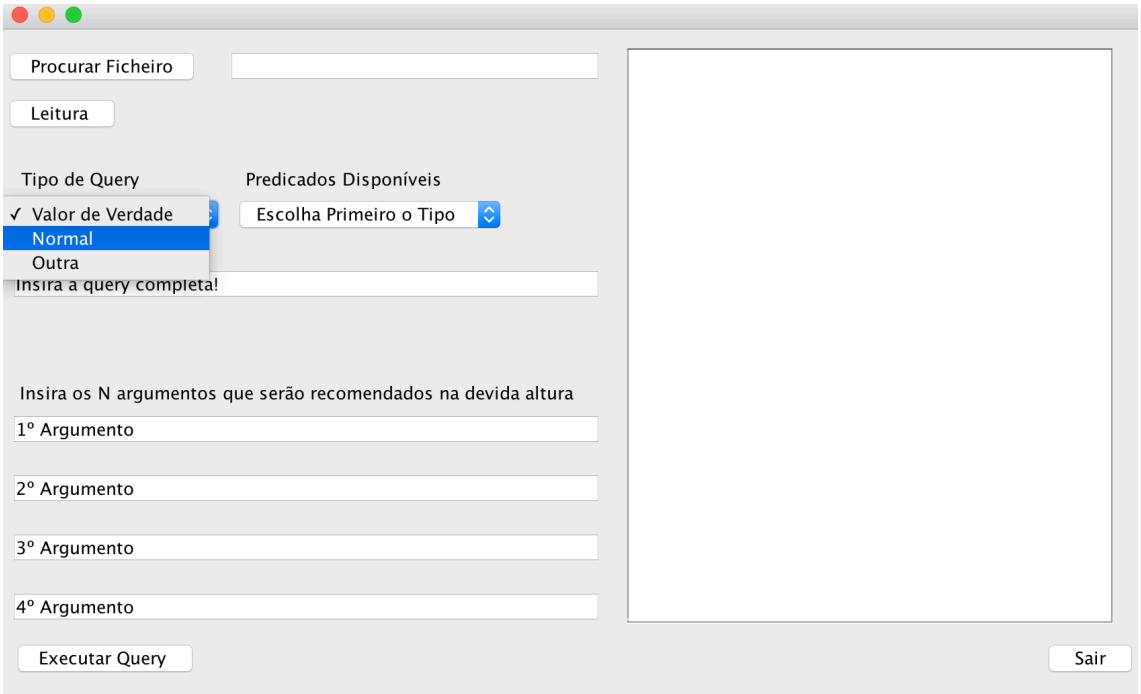

Figure 3 - Escolha Tipo de Query

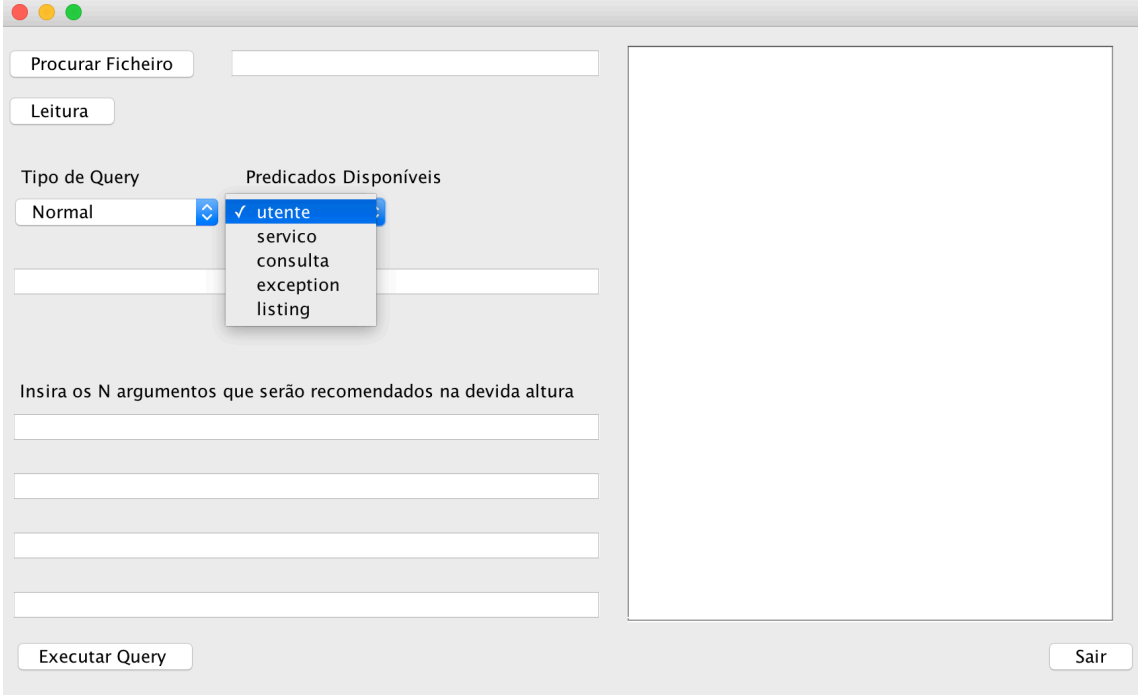

Figure 4 - Query Tipo Normal

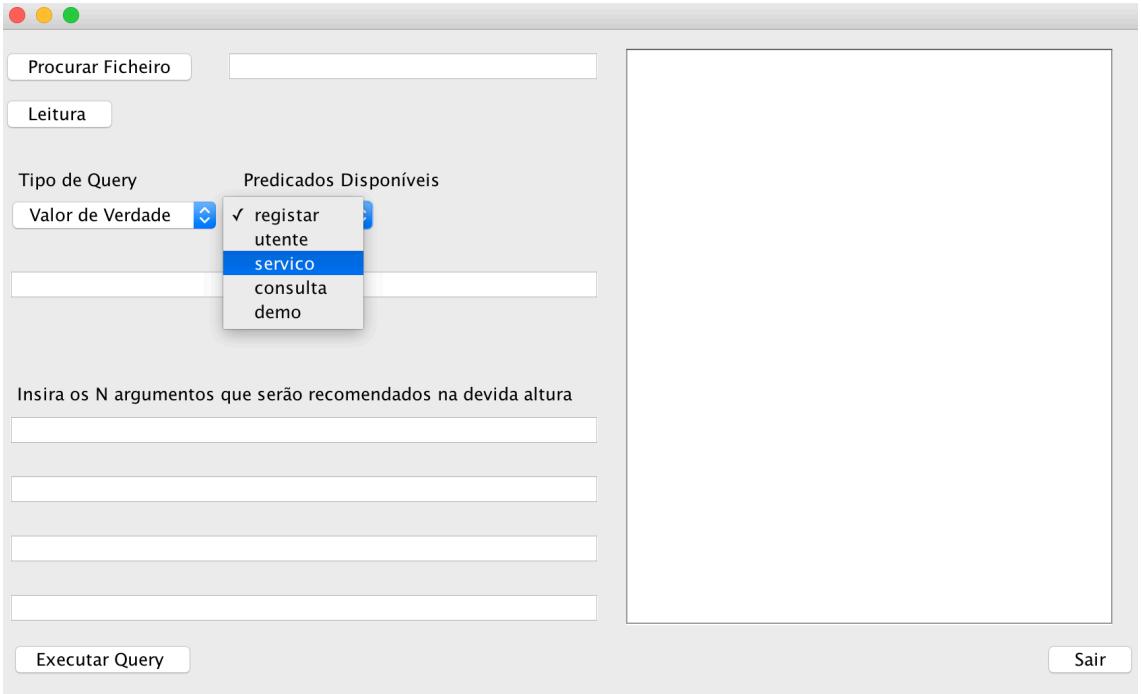

Figure 5 - Query Tipo Valor de Verdade

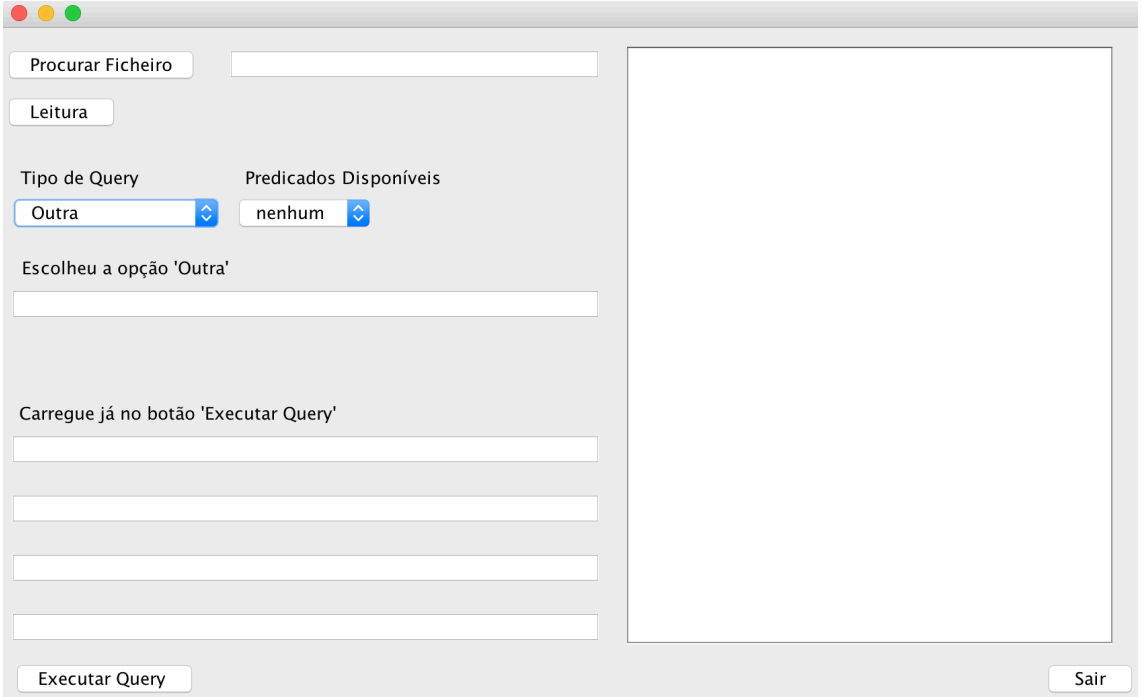

Figure 6 - Query Tipo Outra

Esta interação entre o *JAVA* e o *PROLOG*, do *SICStus* é conseguida com a utilização de métodos existentes na biblioteca JASPER que permite ao sistema realizar as queries da mesma forma que se o fizesse na linha de comandos, e como tal é necessário explicar esses métodos de um forma breve. Esses métodos são *openPrologQuery* e *query*. O primeiro recebe como argumentos uma string com a query a executar e um HashMap onde se guardará a solução calculada, sendo efetuado um processamento ao conteúdo desse por forma a que a sua apresentação possua um refinamento. Este método é utilizado em queries do tipo 'Normal' , isto é, para queries em que se pretenda ter acesso a conteúdo inserido na base de conhecimento. O outro método é o mais indicado para a obtenção de respostas que indicam os valores de verdade, de acordo com o sistema de inferência utilizado, neste caso, recorre-se ao demo, cujo sistema de inferência é caracterizado por três valores de verdade, 'falso', 'verdadeiro' e 'desconhecido', visto que devolve um booleano e recebe como argumentos os mesmos que a anterior.

Para terminar esta análise apenas é necessário mencionar que para qualquer query efetuada as respostas serão sempre apresentadas no campo em branco que ocupa a maior parte do lado direito da janela e sempre que efetua uma nova query o resultado anterior desaparece da mesma com o intuito de facilitar a leitura do resultado.
Também é de realçar que a mesma efetua um controlo dos argumentos, ou seja , se um predicado necessitar de quatro argumentos e o utilizador não preencher os campos indicados será devolvido um aviso referente á situação de existência de campos vazios. E no caso de o utilizador escolher a opção 'Normal', os argumentos terão de possuir letras maiúsculas, uma vez que essas são variáveis e não atributos.

Na opção ‘Outra’ o mesmo não acontece, ou seja, o utilizador com esta opção poderá efetuar a interação da mesma forma que o faria pela linha de comandos disponibilizada pelo *SICStus*. Todas as respostas foram sujeitas a um *parsing* por forma a apresentar as mesmas ao utilizador de uma forma mais refinada em relação á forma como a biblioteca *JASPER* o disponibiliza.

5. Conclusão

Neste trabalho, o principal objetivo foi a construção de um novo sistema de inferência que permita aceitar o desconhecido. Foi incorporado para tal um novo predicado demo, bem como o predicado demoExtendido que permite ao utilizador 'enviar' mais que uma questão, mas apenas utilizando um único atributo. Como base foi novamente utilizada a linguagem de programação em lógica estendida - *PROLOG* - para a representação de conhecimento, bem como para o tratamento da problemática da evolução do mesmo. A lógica estendida deixa de possuir os pressupostos de mundo fechado e de domínio fechado, e agrega o pressuposto de mundo aberto. Isto é, agora o conhecimento pode tomar três valores de verdade, {Verdade,Falso, Desconhecido}.

Para a produção dos predicados extra aproveitou-se o conceito adotado no exercício prático anterior, num sentido mais estatístico, como é o caso da média das idades.

Foi também pedido que a interação entre o utilizador e o sistema fosse efetuado através de uma interface de modo a tornar mais agradável a mesma em relação á habitual linha de comandos. Para tal recorreu-se á linguagem *JAVA*, mais concretamente, á componente *SWING* que permitiu elaborar uma interface de utilização simples e ao mesmo tempo com diferentes funcionalidades. Mas para se conseguir extrapolar dados bidireccionalmente foi necessário utilizar uma libraria, *JASPER*, que vem com o *SICStus* e que possui uma *API* que nos permitiu realizar *queries* através da interface gráfica produzida e obter as devidas respostas produzidas pelo *SICStus* da mesma forma que obteríamos caso se estivesse a utilizar a linha de comandos disponibilizada para tal efeito. A interface não apresenta nenhum problema no que diz respeito á comunicação bidirecional entre as duas plataformas.

Como pudemos verificar ao longo das análises de resultados que foram feitas neste trabalho, pode-se concluir que o sistema funciona sem qualquer tipo de problema, o que nos deixa a nós, enquanto grupo, bastante satisfeitos pelo produto final que conseguimos obter.