"WEXSYS - Web Expert System"

Por

António Jorge de Almeida Vaz

Dissertação de Mestrado

Instituto Superior de Engenharia do Porto

Porto, Novembro de 2010

Instituto Superior de Engenharia do Porto
Departamento de Engenharia Informática

António Jorge de Almeida Vaz

"WEXSYS - Web Expert System"

Trabalho apresentado ao Departamento de Engenharia Informática do Instituto Superior de Engenharia do Porto como requisito parcial para obtenção do grau de Mestre em Engenharia Informática com a área de especialização "Tecnologias do Conhecimento e Decisão".

Orientador:
Doutor Luiz Felipe Rocha de Faria
Júri:
Doutor José António Reis Tavares (Presidente)
Doutor Nuno Filipe Teixeira Malheiro
Doutor Luiz Felipe Rocha de Faria

Porto, Novembro de 2010

Agradecimentos

Gostaria de agradecer a todas as pessoas que, directa ou indirectamente, ajudaram na realização deste trabalho.

Começo por agradecer ao meu orientador **Dr. Luiz Faria** pela disponibilidade que sempre demonstrou e pelas várias conversas construtivas que tivemos.

À **Dra. Fátima Rodrigues** pela compreensão demonstrada.

Ao meu antigo colega e chefe de sector **António Pinto** pela cedência do trabalho que deu origem a esta dissertação.

Ao meu colega de curso **António Mota** pelo companheirismo, pela amizade e em especial por todo o apoio.

Aos meus amigos, em especial ao pessoal dos **Kruppa**, pela amizade e pelo incentivo que sempre me deram.

À minha **família** pela compreensão e pela força sempre demonstradas.

Por fim, quero agradecer a uma pessoa muito especial, à **Clara**, pela compreensão, incentivo e carinho nos momentos mais importantes.

Resumo

Desde o início que os sistemas periciais não foram vistos apenas como sistemas que poderiam substituir os peritos. Os peritos por sua vez, independentemente da área em que operam, eram tidos como indivíduos que atingiram a excelência através da experiência, estudo e total dedicação, por vezes durante anos.

Hoje, mais do que assumir o papel de um perito na excelência de uma área de actuação, os sistemas periciais sobressaem pela disponibilidade contínua, acessos facilitados, custos reduzidos, estabilidade de funcionamento e coerência de raciocínio. Empurrados pelos avanços actuais das redes em termos de velocidade e propagação global, estes sistemas são hoje disponibilizados de forma mais simples e acessível.

Este trabalho é realizado com o propósito de evoluir o actual sistema para uma versão mais apelativa, funcional e fiável. Um dos objectivos passa pela evolução do actual modo de funcionamento, utilização local e apenas um utilizador, para um modo de funcionamento que permita uma disponibilização num ambiente de acesso global, acessível a qualquer hora e em qualquer local.

Palavras-chave: GISPSA, sistema pericial, manutenção base de conhecimento, análise gráfica de base de conhecimento,

Abstract

From the begining that the expert systems were not treated solely as systems intended to replace experts. Experts in turn, regardless of their operating area, were seen as individuals that achieved excellence through experience, study and total dedication, sometimes over many years.

Today, more than just assuming the part of an expert in a given area, expert systems stand for their continuous availability, easy access, reduced costs, operating stability and consistency of reasoning. Pushed by todays advances in networks regarding speed and global spread, these systems are now available in a more simple and accessible way.

This work is carried out in order to evolve the current system to a more appealing, functional and reliable. One of the objectives is the evolution of the current mode of operation, local and single user, to a mode globally accessible regardless the time or the place where its intervention is necessary.

Keywords: GISPSA, expert system, knowledge base maintenance, knowledge base graphical analysis.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Código

Lista de Algoritmos

Lista de Acrónimos

AI *Artificial Intelligence* ou Inteligência Artificial

BRMS *Business Rule Management System*

CF *Certainty Factors* ou Factores de Certeza

CLIPS *C Language Integrated Production System*

DTO *Data Transfer Object*

ES *Expert System*

GIS Gestão Integrada de Seguros

GISPSA GIS Problem Solver Advisor

IDE *Integrated Development Environment*

JEOPS *Java Embedded Object Production System*

JSF *JavaServer Faces*

LHS *Left-Handed Side*

LOMA *Landfill Operation Management Advisor*

MDSS *Medical Diagnosis Support System*

ODBC *Open Data Base Connectivity*

OO *Object Oriented*

RHS *Right-Handed Side*

SP Sistema Pericial

SDK *Software Development Kit*

1

Introdução

Este trabalho é desenvolvido no âmbito do Mestrado em Engenharia Informática, ramo de Tecnologias do Conhecimento e Decisão e teve a sua origem a partir de um sistema, doravante designado por sistema base, desenvolvido por Pinto. Este sistema base foi desenvolvido durante a realização de um trabalho académico (Pinto (1998)), cujo principal objectivo era a construção de um sistema pericial.

Na altura da sua criação, o sistema base apenas estava preparado para uma utilização do tipo mono-posto. Ficava assim estabelecida a condição de apenas um utilizador de cada vez ter acesso ao sistema base. Não era exigível nem sequer necessário que outro cenário fosse equacionado dado tratar-se de um trabalho académico. O sistema ficou, desta forma, limitado.

Assim, somos remetidos a um dos objectivos iniciais que levaram ao desenvolvimento do trabalho que agora aqui se apresenta. Proceder à evolução do sistema base garantindo que o mesmo possa ser acedido por múltiplos utilizadores de forma simultânea e a sua disponibilização num ambiente de acesso global, mediante a utilização de redes, sendo que a preferência recai sobre a rede mais global que existe: a internet. Referindo-nos a este tipo de disponibilização, com todas as vantagens que lhe estão afectas, o próximo passo lógico seria a integração do novo sistema num portal.

É também um objectivo inicial proceder à alteração da actual forma de manutenção do conhecimento. Pretende-se desenvolver um mecanismo de manutenção de conhecimento que simplifique a interacção com o utilizador.

Por último, os objectivos deste trabalho passam igualmente pelo melhoramento do sistema base, tentando eliminar alguns erros que foram detectados ao longo do tempo e acrescentando algumas características que entretanto surgiram e que foram consideradas necessárias.

1.1 Enquadramento

O sistema a desenvolver, embora tenha como ponto de partida um sistema base criado para utilização mono-posto, fará parte de um portal onde será integrado como uma ferramenta de apoio na resolução de problemas operacionais.

Para além de, actualmente, fazer muito mais sentido a disponibilização do sistema num ambiente de acesso globalizado e com a possibilidade de vários utilizadores em simultâneo, a grande vantagem reside no facto de poder utilizar a mesma base de conhecimento que alimenta o sistema pericial independentemente do local onde se realizem os acessos. Fica assim garantida a uniformidade, a fiabilidade e a estabilidade do sistema.

Por último, a globalização do sistema garante igualmente que, sempre que este é utilizado, não para consulta ou apoio na resolução de problemas, mas para acrescentar novos factos e dados à base de conhecimento, estes estejam imediatamente disponíveis para novas consultas ou para resoluções de problemas de forma mais eficaz.

1.2 Objectivos

O objectivo principal deste trabalho passa pela integração do GIS Problem Solver Advisor (GISPSA) (sistema base) num portal. Além deste, outros objectivos podem ser considerados como de eventual ou relevante importância podendo ser caracterizados da seguinte forma:

- Objectivos globais
  - Desenvolvimento de um sistema pericial ⇒ Pretende-se desenvolver um candidato à substituição do GISPSA;
  - Manutenção da base de conhecimento ⇒ Gerir a base de conhecimento relativamente à inserção e actualização de regras de conhecimento, bem como implementação de mecanismos de verificação das regras;
- Objectivos específicos
  - Adaptar o motor de inferência existente ou desenvolver novo ⇒ Desenvolvimento de um motor de inferência que se enquadre com os objectivos definidos;
  - Alterar o mecanismo de interacção do utilizador com a base de conhecimento ⇒ Facilitar a manipulação da base de conhecimento de forma a evitar problemas na alteração da base de conhecimento.

- Verificar e detectar inconsistências na base de conhecimento ⇒ Implementar mecanismos de verificação das regras existentes para minimizar problemas;

1.3 Planeamento

O planeamento deste trabalho foi estruturado em quatro fases. A composição dessas fases é a seguinte:

1. Requisitos
   - Levantamento de requisitos
   - Estado da arte
   - Ferramentas existentes
2. Protótipo
   - Protótipo do motor de inferência
   - Implementação do gestor da base de conhecimento
3. Verificação
   - Implementação de métodos de verificação e detecção de inconsistências
4. Elaboração da dissertação
   - Redacção final da dissertação

1.4 Estruturação da dissertação

Nesta secção é apresentada a estrutura da dissertação composta por cinco capítulos.

No primeiro capítulo, "Introdução", é apresentado o problema actual, quais os objectivos a alcançar e qual o planeamento para a prossecução desses objectivos.

No segundo capítulo, "Estado da Arte", é apresentado a solução GISPSA e também o estado da arte na área dos sistemas periciais.

No terceiro capítulo, "Ferramentas utilizadas", é apresentado o estudo efectuado sobre as ferramentas utilizadas e quais as conclusões do mesmo.

No quarto capítulo, "Wexsys", é descrita a solução proposta e implementada.

No quinto e último capítulo, "Conclusões", são apresentadas as conclusões finais bem como os objectivos alcançados e as actuais limitações.

2

Estado da arte

Num estado da arte, pretende-se essencialmente fazer o enquadramento do estágio de evolução e eventuais casos de estudo relacionados com as tecnologias, sistemas ou propostas que são utilizadas ou apresentadas ao longo de um trabalho.

Para o nosso caso em concreto, vamos focalizar esse enquadramento sobre os sistemas periciais, no seu sentido mais abstracto, e ainda sobre o sistema base já indicado, o GISPSA. Apesar de o sistema base ser, por si só, um sistema pericial e como tal poder ser apresentado como um caso de estudo inserido nos sistemas periciais generalizados, o facto de servir como ponto de partida para este trabalho em concreto, confere-lhe uma importância muito superior, sendo quase obrigatória a reserva de uma secção própria para o mesmo.

2.1 Sistemas periciais

Os sistemas periciais pertencem a um dos ramos da *Artificial Intelligence* ou Inteligência Artificial (AI) e surgiram a partir da necessidade que alguns investigadores sentiram em criar um *software* que demonstrasse raciocínio e comportamento inteligente durante a resolução de problemas.

Um Sistema Pericial (SP), também conhecido como *Expert System* (ES), caracteriza-se como sendo um programa que permite a consulta e utilização de conhecimento extraído de um ou mais peritos em determinada área. Esse conhecimento está normalmente guardado naquilo que é mais habitualmente designado por base de conhecimento e à qual o programa tem o devido acesso.

Na criação de um SP, o ideal será sempre consultar os melhores peritos dentro da área pretendida e ainda que estes estejam disponíveis para partilhar e disponibilizar o seu conhecimento. Tal seria, efectivamente, a situação ideal mas que nem sempre se

aproxima da realidade. Aliás, o verdadeiro sucesso de um sistema pericial reside na qualidade da sua base de conhecimento.

Antes de mais, devemos considerar que alguém que é perito em algo, certamente o será por uma série de factores conjugados. Experiência adquirida ao longo de vários anos, formação específica, dedicação e estudo profundo, serão alguns desses factores. Sempre que um indivíduo se encontra numa posição de excelência, em que é considerado um verdadeiro perito, encontra-se também numa posição em que lhe é bastante difícil encontrar o tempo necessário para se submeter a uma extracção do conhecimento que detém. Se a isto juntar-mos o facto do objectivo, pelo qual se procede à extracção de conhecimento, é a criação de um programa que será equiparado ao próprio perito em termos de conhecimento certamente que haverá algum desconforto.

Aqui surgem as primeiras dúvidas. Será que o sistema pericial tomará o lugar de um ou mais peritos? Apesar desta e de outras preocupações, os peritos normalmente contribuem de forma positiva para o enriquecimento de um sistema pericial tornando-o, desta forma, num sistema com maior qualidade e do qual eles próprios poderão usufruir já que, quando é possível agregar o conhecimento de múltiplas pessoas, obtém-se um conhecimento global superior ao individual.

Para além disto, o sistema pericial tende a proceder a um raciocínio mais linear, lógico e coerente, quando comparado com um ser humano que poderá deixar que factores como por exemplo o cansaço ou os sentimentos o influenciem.

Merritt define um sistema pericial como sendo um programa que utiliza conhecimento e métodos de inferência para resolver problemas que são suficientemente complexos para requerer um especialista para a sua resolução:

"... an intelligent computer program that uses knowledge and inference procedures to solve problems that are difficult enough to require significant human expertise for their solutions" Merritt (1989)

A definição de Giarratano *et al.* é mais relacionada com o tipo de problemas tratados por um sistema pericial:

"... computer application wich embody some non-algorithmic expertise for solving certain types of problems." Giarratano *et al.* (1998)

Esta definição enquadra, de uma forma global, as definições reconhecidas aos sistemas periciais: "um sistema pericial pretende emular um perito na identificação de um determinado problema, cuja resolução envolva conhecimento não algorítmico".

Existe uma clara separação entre conhecimento e raciocínio, o conhecimento é obtido, como já foi referido, recorrendo a um ou mais peritos na área. Os dados obtidos são depois

codificados através de um determinado esquema de representação. No passo seguinte, é adicionada a capacidade de raciocínio ao sistema pericial recorrendo a estratégias heurísticas na tentativa de fornecer um resultado semelhante ao que um perito forneceria e aqui, uma vez mais, o papel do perito é fundamental para o sucesso do sistema pericial.

Para além de contribuir com o conhecimento, o perito deverá igualmente contribuir com o raciocínio utilizado para o manuseamento desse mesmo conhecimento. Isto porque a resolução de um problema poderá, em determinadas situações, necessitar de pequenos ajustes que fogem à linearidade de um processamento computacional mas que podem perfeitamente ser simuladas e recriadas na construção do raciocínio do programa.

É necessário compreender que um perito, para além de ser alguém que é considerado um especialista na sua área, é habitualmente alguém com pouco ou nenhum tempo disponível, o que dificulta bastante a realização da recolha do conhecimento. O seu tempo é dispendioso e tem por norma poucos espaços de manobra devido às muitas solicitações que lhe são dirigidas. Aqui impera o factor de resistência, ora porque o perito não acredita que o sistema seja tão eficaz como ele, ora porque existe o receio de ser substituído ou passar a ter menos solicitações. Quanto maior a colaboração do ou dos peritos envolvidos na criação de um qualquer sistema pericial, melhor e mais eficaz será esse sistema pericial.

2.1.1 Arquitectura

As duas grandes componentes do sistema pericial e que permitem a separação referida entre conhecimento e raciocínio, são a base de conhecimento (depende do domínio ou da área abrangida) e o bloco de raciocínio (não depende do domínio). Este último é ainda dividido em três partes distintas: o motor de inferência, o sistema de explicações e a interface com o utilizador.

Um sistema pericial deve ser estruturado de forma a que possa ser actualizado facilmente sem que seja necessária qualquer alteração ao nível funcional do programa.

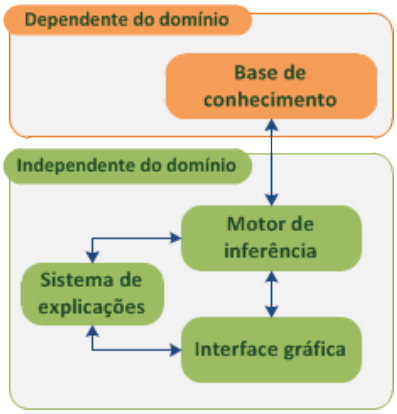

Figura 1 Arquitectura de um sistema pericial

A figura 1 mostra a composição de um sistema pericial e que, mais detalhadamente, é a seguinte:

- Interface com o utilizador:

  um sistema pericial deverá permitir a interacção com o utilizador. Nos casos mais habituais, os utilizadores que usufruem dos benefícios destes sistemas, não são peritos da área e pretendem aconselhamento ou ajuda na resolução de um problema. Noutros casos, o utilizador poderá ser um perito da área que apenas pretende testar o sistema pericial ou confrontar as conclusões deste com as suas próprias conclusões para perceber se, sendo diferentes, qual o motivo para que isso aconteça. Isto poderá permitir que um perito aprofunde ou corrija o seu próprio conhecimento ou que, por outro lado, se procedam a correcções à base de conhecimento do sistema pericial mediante se trate de erro por parte de um ou de outro.

  De qualquer forma, a interface com o utilizador deverá permitir a interacção entre o sistema pericial e o utilizador (qualquer que este seja) para que o utilizador responda às questões que lhe são colocadas e peça explicações para os passos e conclusões que percorre e descobre ao longo da utilização do programa.

- Base de conhecimento:

  o conhecimento extraído de um ou mais peritos é armazenado e mantido aqui. Durante a execução do programa, o sistema pericial acede a esta base para consultar os dados disponíveis. É sempre possível, em qualquer altura, retirar, acrescentar ou corrigir factos na base de conhecimento. Isto contribui para uma evolução da mesma no sentido de melhorar sempre o funcionamento do próprio sistema pericial.

- Motor de inferência:

  este bloco é o grande responsável por interpretar o conhecimento e colocar, de uma forma estruturada, questões ao utilizador. O raciocínio extraído do ou dos peritos é aqui consultado para que, mediante as questões que são colocadas ao utilizador e as consultas que são efectuadas à base de conhecimento, este motor de inferência determine qual a próxima questão a colocar, qual a próxima consulta a realizar ou qual a conclusão a apresentar.

- Sistema de explicações:

  sempre que o utilizador pretende que lhe seja dada uma explicação para a questão que está a ser colocada ou para a conclusão que foi determinada, este bloco será o responsável por dar essa mesma explicação.

Esta divisão permite que os módulos funcionais do programa possam ser isolados, como podemos observar na figura 1, de forma a facilitar futuras correcções e actualizações da base de conhecimento, contribuindo positivamente para a evolução da mesma e evitando assim que o sistema pericial fique obsoleto.

2.1.2 Base de conhecimento

Uma base de conhecimento é um conjunto de informação representada de forma a que possa ser processada pelo motor de inferência. Existem várias formas de representação de conhecimento, no entanto, as mais conhecidas são a representação por regras, representação por "frames" e ainda as redes semânticas.

Ao representar conhecimento, um dos factores determinantes para o método necessário a essa representação é o domínio do problema ou área a modelar. A título de exemplo, se um problema segue um padrão ou algum tipo de linearidade na busca da solução necessária, ou ainda se a solução puder ser obtida através de questões cujas respostas são maioritariamente simples (sim, não, valores numéricos, etc), a representação do conhecimento é feita utilizando os métodos de regras ou "frames".

Como já foi referido, o papel de um perito não está limitado apenas à contribuição com o conhecimento para a base necessária. Este exerce um outro papel igualmente importante que tem a ver mais com o aspecto de raciocínio. Desta forma, ao consultar a base de conhecimento, são tomadas algumas decisões tais como, verificação de eventuais soluções já possíveis em pontos intermédios ou qual a próxima questão a colocar. Este método de interacção com a base de conhecimento poderá eventualmente entrar em

conflito com a independência de domínio que foi referida na arquitectura de um sistema pericial. No entanto, este aspecto fundamental da criação de um sistema pericial contribui largamente para o bom funcionamento do mesmo.

Para além disto, a base de conhecimento deverá sempre contemplar as funcionalidades necessárias que permitam ao utilizador, em qualquer altura, proceder à adição de novos factos, à correcção dos já existentes, ou ainda à eliminação dos mesmos. Isto permite manter uma base de conhecimento sempre actualizada, fiável e consistente. Estas razões contribuem para a resolução cada vez mais eficaz dos problemas que são colocados. Para novos problemas que surgem, poderão ser necessárias novas soluções ou novos métodos para as encontrar. Para problemas já conhecidos, poderão ser encontradas novas formas ou novas soluções de os resolver, eventualmente consideradas mais eficazes ou de implementação mais simples.

Por tudo isto considera-se que a base de conhecimento, apesar de não ser o verdadeiro motor de um sistema pericial, poderá ser encarada como o verdadeiro combustível que o move e sem o qual não seria possível qualquer tipo de evolução pelo que a não contribuição para o seu melhoramento estagna o próprio sistema, provocando, mais cedo ou mais tarde, o abandono do mesmo por parte dos utilizadores que entretanto podem perder a confiança que lhe depositaram.

Os métodos de representação de conhecimento podem ser seleccionados de acordo com as características que lhes estão inerentes e que se descrevem de seguida.

2.1.2.1 Regras

Uma das técnicas de representação de conhecimento mais utilizadas, é a representação por regras. Estas são utilizadas para representar heurísticas e caracterizam-se por um conjunto de premissas que desencadeiam ou conduzem a uma solução.

As regras têm o formato: <identificação>: Se <*Left-Handed Side* (LHS)> Então <*Right-Handed Side* (RHS)>

Onde:

<identificação> - identificação da regra.

<LHS> – "Left-Handed Side" define as premissas que terão que ser satisfeitas de forma a que a regra possa ser disparada.

<RHS> – "Right-Handed Side" define as acções ou conclusões que serão verificadas com o disparo da regra.

As premissas são definidas recorrendo a factos que podem ser previamente conhecidos ou inferidos a partir de outras regras. A utilização de regras promove uma separação

entre a parte lógica e a funcional, isto é, permite-nos uma abstracção de implementação focando-se na lógica.

Vantagens:

- expressividade - as regras são normalmente representadas de uma forma natural que pode mesmo chegar a ser declarativa. É perfeitamente normal uma regra estar escrita num vocabulário que pode ser lido e compreendido por alguém que não seja o programador, o perito ou alguém com formação na área em questão;
- modularidade - as regras são independentes entre si o que possibilita uma mais fácil manutenção da base de conhecimento. Isto permite e facilita a remoção de regras, a adição de outras ou a alteração das já existentes;
- separação lógica - as regras são independentes do motor de inferência e este não carece de alterações ou actualizações quando as regras são modificadas. O funcionamento do sistema não é alterado quando as regras o são, apenas as soluções que o sistema obtém é que poderão ser diferentes (normalmente para melhor);
- gramática rígida - as regras obedecem a estruturas que estão perfeitamente definidas à partida. Isto implica que as mesmas possam ser validadas ou até mesmo construídas recorrendo a outros sistemas;
- execução lógica - é sempre possível, em qualquer altura do funcionamento do sistema pericial, solicitar o desencadeamento ou disparo de uma sucessão de regras. A isto chama-se uma explosão combinatória das regras e permite que sejam contemplados vários cenários possíveis utilizando apenas uma execução. Esta execução lógica tem sempre o mesmo resultado desde que se mantenham os parâmetros iniciais;
- sistema de explicações - ao proceder a disparos de regras, é possível dar indicação de qual a lógica utilizada para disparar uma determinada regra. Isto permite, de uma forma didáctica e simples, justificar a razão dos disparos auxiliando o utilizador na compreensão da evolução que o sistema segue para atingir determinada solução.

Desvantagens:

- explicações - se um utilizador não conhecer as regras ou não as aceitar o sistema de explicações não será uma mais valia. Poderá até eventualmente tornar-se em algo que confunde mais do que aquilo que auxilia;

- gramática rígida - apesar de o método de utilização de estruturas poder ser considerado importante porque estabelece um padrão a ser seguido, pode conduzir a problemas de expressividade quando utilizado em domínios complexos;

- inconsistência - um dos maiores problemas que podem surgir num sistema que utiliza a representação do conhecimento através de regras, tem a ver com a possibilidade da existência de inconsistências e incoerências nas regras. Isto reduz-se ou evita-se através da utilização exaustiva do sistema numa fase de testes, em que são equacionados o maior número de casos ou cenários que possam despoletar todas as regras existentes na base de conhecimento.

2.1.2.2 Frames

Uma frame representa um conceito e é composta por *slots* que que não são mais do que a definição de um conjunto <atributo, valor>. Um valor pode ser um elemento único ou uma lista de valores e em termos de dados pode conter um tipo de dados primitivo ou uma referência a outra frame.

Esta representação tem uma ligação muito forte com o paradigma da programação orientada a objectos já que a ideia é a mesma: representar objectos e conceitos do mundo real utilizando a descrição, a composição e a herança.

Como vantagens temos:

- associação - as frames definem e representam o conhecimento de uma forma associativa (associação normal de conceitos).

Desvantagens:

- conhecimento prévio - esta forma de representação de conhecimento exige que todo o espaço do problema seja conhecido de uma forma exaustiva por parte dos utilizadores. Não é muito fácil para um utilizador que não esteja envolvido de forma profunda na área em questão utilizar de forma eficaz o sistema pericial nestes moldes;

- alargamento do espaço - as frames podem representar um espaço que não está restringido ao domínio do problema. Isto conduz a dificuldades acrescidas na utilização do sistema;

2.1.2.3 Redes semânticas

Uma rede semântica é um modelo formal em que são descritos conceitos e relações recorrendo a um grafo dirigido. Os conceitos são representados por nós, as relações por arcos e existe uma herança de conceitos à medida que se procede à navegação pelas várias relações.

Como vantagens temos:

- exploração gráfica - a representação de conhecimento através de redes semânticas permite a um utilizador explorar um domínio graficamente. Isto facilita a interacção e utilização do sistema;

- facilidade na adição de clusters relacionados - utilizando o modo gráfico de representação, torna-se mais fácil adicionar novos clusters em que existam conceitos relacionados;

- fácil assimilação - os seres humanos relacionam-se bem com ambientes gráficos e têm uma tendência a não só aceitá-los melhor como também a proceder a uma utilização mais atractiva, simples e de rápida aprendizagem com um mínimo de formação.

Como desvantagens temos:

- conhecimento prévio - esta forma de representação de conhecimento exige que todo o espaço do problema seja conhecido de uma forma exaustiva por parte dos utilizadores. Não é muito fácil para um utilizador que não esteja envolvido de forma profunda na área em questão utilizar de forma eficaz o sistema pericial nestes moldes;

- alargamento do espaço - as redes semânticas podem representar um espaço que não está restringido ao domínio do problema. Isto conduz a dificuldades acrescidas na utilização do sistema;

- falta de estrutura e formalidade - não existe uma semântica formal para a representação simbólica das ligações o que deixa demasiado espaço para a criatividade e eventualmente para o surgimento de problemas associados ao facto de poderem existir múltiplas formas de representação.

2.1.2.4 Mecanismos de inferência

A base de conhecimento representa o caminho para encontrar uma determinada solução mas o motor de inferência é a entidade que efectivamente percorre esse caminho. Um motor de inferência pode utilizar diferentes estratégias de raciocínio para a obtenção de respostas. As mais conhecidas são a estratégia de encadeamento directo e a de encadeamento inverso. Contudo, é possível combinar as duas no mesmo sistema, naquilo que é vulgarmente conhecido como uma estratégia de encadeamento misto.

Encadeamento directo (*Forward Chaining*) – Um ser humano, no seu dia a dia, utiliza esta estratégia para processamento de raciocínio. Consoante assistimos a acontecimentos, o cérebro processa-os e analisa-os decidindo o que fazer de seguida. De igual modo, a estratégia de raciocínio por encadeamento directo começa por percorrer os factos básicos que vão originar o disparo de regras, o que levará à obtenção de conclusões intermédias que, por sua vez, irão originar o disparo de mais regras e assim sucessivamente. O processo continua até que se obtenham conclusões finais, se tal for possível, ou não haja mais lugar à possibilidade de disparo de novas regras.

Devido às suas características, esta estratégia é mais apropriada para: planeamento, monitorização, controlo e interpretação.

Encadeamento inverso (*Backward Chaining*) – No encadeamento inverso, o motor de inferência tenta provar as conclusões finais que aparecem no lado direito das regras (RHS). Essas conclusões finais também são designadas como objectivos e é normal dizer-se que o motor de inferência com encadeamento inverso é orientado aos objectivos (*goal driven*).

Para provarmos uma dada conclusão, teremos que provar as condições que aparecem no lado esquerdo da regra (LHS), que poderão ser suportadas por conclusões intermédias de outras regras ou por factos básicos. Sendo assim, o mecanismo de encadeamento inverso assume um carácter essencialmente recursivo (para provar uma conclusão vamos ter que provar as condições).

Este tipo de estratégia é mais apropriada para diagnósticos já que temos uma hipótese que necessita de ser provada de forma a chegar a uma conclusão.

É importante salientar que a forma de representação das regras é independente da estratégia utilizada pelo motor de inferência. No entanto, é possível melhorar a eficácia do sistema se adaptarmos essa representação consoante o encadeamento utilizado.

2.1.2.5 Manutenção de uma base de conhecimento

A manutenção de uma base de conhecimento, embora seja considerada vital para a sobrevivência do sistema pericial, é um processo bastante complexo no sentido em que implica cuidados que nem sempre são tidos ou considerados. É habitual existirem problemas de validação, verificação, redundância e contradições.

De forma a garantir que a base de conhecimento se mantenha fiável e bem definida devem ser desenvolvidos mecanismos que evitem a ocorrência deste tipo de problemas.

Redundância - Considera-se aplicação de redundância a uma base de conhecimento quando, ao inserir uma regra, estamos a adicionar conhecimento que já estava contido na base de conhecimento. Podemos observar o seguinte exemplo:

regra1: **se** X nada **então** X é_peixe.

regra2: **se** X tem_barbatanas **então** X nada.

regra3: **se** X nada **E** X tem_barbatanas **então** X é_peixe.

Neste exemplo, é possível verificar que esta base de conhecimento mantém-se idêntica (o conhecimento é o mesmo) se removermos a regra3, considerada redundante. Esta regra não trouxe nenhum conhecimento novo ao sistema e pode ser descartada.

Contradição - Considera-se regra contraditória, toda a regra que contradiz conhecimento já existente na base de conhecimento. Por exemplo:

regra1: **se** X nada **então** X é_peixe.

regra2: **se** X nada **então** X não_é_peixe.

A regra2 contradiz claramente a regra1 pelo que é considerada contraditória. Um dos grandes problemas na análise de duas ou mais regras contraditórias, passa por determinar quais as que correspondem ao conhecimento correcto.

Verificação - acção desencadeada ainda na fase de implementação inicial da base de conhecimento, onde se pretende minimizar problemas essencialmente de redundância e de contradição que possam ser provocados pelo programador. Toda a base de conhecimento é verificada e confrontada com as notas, apontamentos e indicações a que o programador teve acesso e que serviram de alicerce para a construção da mesma.

Validação - a acção preventiva de erros iniciais levada a cabo logo após a implementação da base de conhecimento, precede a acção seguinte que consiste em validar a referida base através da intervenção de um perito, o qual utiliza vários cenários de testes onde tenta obter as respostas ao mesmo tempo que vai consultando as explicações disponíveis ao longo do percurso.

Apesar de ser habitual proceder a esta validação com o perito que contribuiu com o conhecimento, é igualmente habitual e até benéfico utilizar outros peritos da mesma

área que, para além de validarem a base de conhecimento em causa, poderão inclusive confrontar o seu próprio conhecimento com o do perito original, procedendo a correcções a eles próprios ou à base *per si*. Quanto maior a taxa de intervenção construtiva por parte de peritos numa base de conhecimento, melhor será a qualidade associada à mesma.

2.1.3 Raciocínio sobre incerteza

Raciocínio sobre incerteza é o nome dado ao processo de formulação de conhecimento, baseado em fontes de informação que detenham algum tipo de incerteza na sua especificação.

O conhecimento que detemos não se limita a observações lógicas do tipo "se Observação_A então conclusão_B". Em várias situações necessitamos de lidar com informação incompleta, incorrecta ou até mesmo ausente.

Quando lidamos com conhecimento de peritos num determinado domínio surge ainda outro tipo de conhecimento, este originado através da observação e experiência.

Existe então a necessidade de representação de conhecimento e respectivo grau de certeza. Para este efeito existem algumas técnicas, tais como: lógica *Fuzzy*, teorias de probabilidades tais como *Bayes* ou *Dempster-Shafer*, factores de certeza, entre outros.

Os seguintes autores podem ser consultados para referência das técnicas acima mencionadas:

- Lógica *Fuzzy* - Drakopoulos (1994);
- Bayes - Lucas (2001);
- Dempster-Shafer - Wilson (2000) e Barnett (1991);

De seguida vai ser descrita a técnica dos factores de certeza que foi a técnica adoptada para o desenvolvimento da solução.

2.1.3.1 Factores de certeza

A técnica dos *Certainty Factors* ou Factores de Certeza (CF) é um do métodos de tratamento de incertezas num sistema pericial. A técnica foi desenvolvida por Shortliffe and Buchanan para o projecto MYCIN, um sistema pericial de diagnóstico e tratamento de meningite e tratamento do sangue. Desde então que esta técnica é o padrão seguido no desenvolvimento de sistemas periciais baseados em regras Heckerman (1992).

De um ponto de vista funcional caracteriza-se pela atribuição de um número, o grau de certeza, a factos e regras. Esse grau de certeza é uma medida relativa probabilística que pode ser positiva ou negativa, sendo normalmente usados valores no intervalo de -1 a 1.

A sua definição original baseou-se na seguinte fórmula: $FC[H,E] = MC[H,E] - MD[H,E]$.

O factor de certeza da hipótese H dado uma evidência E é dado pela diferença entre:

- *a medida de certeza de H, dado E* $\rightarrow MC[H,E]$*;*
- *a medida de descrença de H, dado E* $\rightarrow MD[H,E]$*;*

O CF final de uma regra com uma premissa baseada num facto ou noutra regra é dado pela fórmula 2.1.

Quando as premissas estão associadas através do operador conjunção é aplicada a fórmula 2.2.

Quando existe mais do que uma regra a suportar uma determinada conclusão, o CF final é o resultado da fórmula 2.3.

$$CF_{final} = CF_{premissa} \times CF_{regra} \tag{2.1}$$

$$CF_{final} = min\left[CF_{premissa1}, ..., CF_{premissan}\right] \times CF_{regra} \tag{2.2}$$

$$X,Y > 0, CF(X,Y) = X + Y(100\check{}X)/100 \tag{2.3a}$$
$$X\ ou\ Y < 0, CF(X,Y) = X + Y/(1\check{}min(\|X\|,\|Y\|)) \tag{2.3b}$$
$$X,Y < 0, CF(X,Y) = -CF(-X,-Y) \tag{2.3c}$$

2.1.4 Vantagens e desvantagens

Apesar de tudo o que foi descrito, um sistema pericial não representa, de forma alguma, a ferramenta mais apropriada para resolver todos os problemas com recurso a conhecimento que, de outra forma, estaria reservado apenas a peritos das várias áreas. Ao utilizarmos um sistema pericial, estamos sujeitos a aspectos positivos e negativos, cabendo a nós o papel de determinar se quando confrontados com um determinado problema, a utilização deste tipo de ferramenta é a mais apropriada.

Para observar quais as principais vantagens de um sistema pericial, podemos consultar as opiniões partilhadas por Giarratano *et al.* (1998); Co/AJRA (1997):

- Elevada disponibilidade ⇒ um sistema pericial está sempre disponível em termos de tempo e local, pronto a funcionar sob qualquer condição, enquanto que um perito não dispõe dessa total disponibilidade;
- Baixo risco de utilização ⇒ pode ser utilizado em ambientes considerados perigosos e hostis para um ser humano;
- Baixo custo ⇒ o custo de providenciar conhecimento por utilizador é bastante reduzido;
- Durabilidade ⇒ o conhecimento é permanente, ao contrário de um especialista que está sujeito a ausências;
- Conhecimento de vários especialistas ⇒ quando agregado, o conhecimento recolhido a partir de vários especialistas poderá conduzir a resultados mais adequados e considerados melhores quando confrontados com o conhecimento individual e os resultados que daí advêm;
- Capacidade de explicação ⇒ um sistema pericial consegue explicar detalhadamente os passos que levaram à resposta dada, enquanto que um ser humano pode não ter a disponibilidade nem a vontade de explicar todo o processo de raciocínio seguido;
- Emotividade ⇒ um sistema pericial não está sujeito a emoções, o seu funcionamento e as respostas que providencia não são influenciadas pelo meio em que está inserido, como por exemplo situações de *stress* ou situações de perigo iminente (a queda de um avião ou uma central nuclear em risco).
- Não influenciável ⇒ acontecimentos mais actuais têm o mesmo peso na tomada de decisão, evitando assim o fenómeno de "viciação" da base de conhecimento.

No entanto, um sistema pericial comporta também algumas desvantagens das quais se salientam:

- Emotividade ⇒ o facto de não ser emotivo também pode ser visto como uma desvantagem, já que não recorrer a emoções pode ser problemático, por exemplo em situações que necessitem de senso comum;

- Manutenção ⇒ se não for devidamente mantido e actualizado um sistema pericial corre o risco de "envelhecer" no sentido em que deixa de ser um sistema fiável a que possamos recorrer e que corre o risco de abandono;
- Criatividade ⇒ um sistema pericial não tem capacidade de reagir de forma diferente a problemas colocados seguindo sempre uma linha estável de raciocínio;
- Adaptação sensorial ⇒ um sistema pericial não consegue reagir a informação sensorial do meio que o rodeia, entenda-se sensorial como sentidos humanos tais como o cheiro o tacto e a visão.

2.2 GISPSA - GIS Problem Solver Advisor

O Gestão Integrada de Seguros (GIS) é um Sistema de Informação vocacionado para a actividade seguradora, residente na plataforma AS/400 da IBM. Umas das características mais interessantes deste sistema, acabou por ser também um dos grandes proporcionadores de eventuais problemas. O GIS possui uma grande capacidade de configuração o que, sendo muito útil para a adaptabilidade deste a cenários, situações ou utilizadores diferentes, acaba por originar vários problemas operacionais através de inconsistências na própria configuração.

Dada esta facilidade aparente de ocorrência de problemas e a falta de sensibilidade e formação técnica dos utilizadores para os resolver, a equipa de assistência técnica afecta ao GIS estava muitas vezes com sobrecarga de pedidos por parte dos clientes que pretendem estes problemas solucionados. Existiam períodos de especial relevância no que toca a estas sobrecargas, tais como os fechos de ano das empresas.

A partir daqui, surge a ideia e a necessidade da criação do GISPSA com o intuito de aliviar a sobrecarga da equipa de assistência técnica mas essencialmente com o grande objectivo de assegurar que um cliente não fica sem solução para os problemas com que se depara apenas porque não estão técnicos disponíveis. A grande vantagem de um sistema pericial foi e será sempre essa: poder substituir um perito sem ter que enfrentar questões como a disponibilidade, custos ou tempo.

Assim surge o GISPSA que mais não é que um projecto criado por Pinto, no âmbito de um projecto académico, cujo objectivo inicial passava por criar um Sistema Pericial para diagnóstico de problemas operacionais do GIS. Este sistema teve por base o conhecimento e o respectivo raciocínio extraídos de um perito da área Administrativa-Financeira do GIS.

No caso de problemas mais complexos em que o Sistema Pericial não consiga determinar os factores de erro, é remetida a resolução para um técnico da empresa, comunicando esse facto ao utilizador Pinto (1998).

2.2.1 Arquitectura

Sendo um sistema pericial, o GISPSA teria que ser desenvolvido utilizando uma linguagem que tivesse uma série de características. A existência de uma base de conhecimento, a possibilidade de consultar factos, adicionar novos factos ou ainda de remover factos nessa mesma base e aspectos como o backtracking na realização de raciocínio ao inferir sobre esses mesmos factos, não permitia um leque muito alargado na escolha da linguagem de

desenvolvimento.

Desta forma, ficou decidido que o GISPSA seria desenvolvido em prolog (mais especificamente, *winprolog*). Devido à simplicidade de utilização e extensa documentação de apoio, o *LPA/Prolog* foi usado como motor de inferência e o *LPA/Flex* foi usado para estabelecer a declaração de regras.

Como já foi referido anteriormente, o GISPSA é uma aplicação *standalone*, desenhada e desenvolvida para uma execução local em mono-posto e sujeita à obtenção de uma licença de utilização para garantir o seu funcionamento.

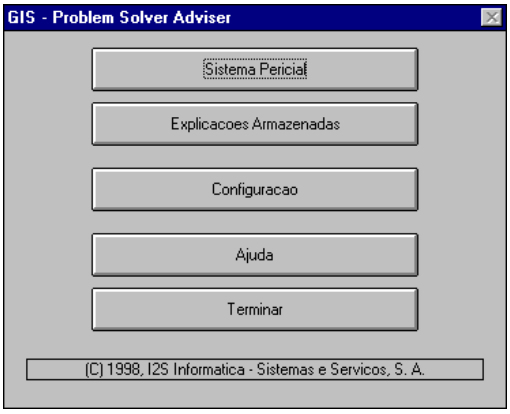

Figura 2 GISPSA menu inicial

A aplicação dispõe de uma interface gráfica também criada em *winprolog* que utiliza recursos específicos do sistema operativo *Windows*.

A base de conhecimento é armazenada num único ficheiro *.ksl* e o seu carregamento não é parcial. O mesmo será dizer que toda a base de conhecimento é passada para memória de trabalho assim que o programa arranca.

2.2.2 Funcionamento

Em termos funcionais, a aplicação permite mudar o tipo de linguagem a ser empregue nas conversações estabelecidas com o utilizador consoante esta esteja a ser utilizada por um técnico (habitualmente os técnicos com formação na área) ou um utilizador normal (o cliente que adquiriu a aplicação e cuja formação é mais reduzida ou até mesmo inexistente).

Após a definição da linguagem de interacção a utilizar, o sistema inicia então o pedido de assistência apresentando a mensagem “Seleccione a aplicação” (figura 3) que permite numa primeira instância direccionar o raciocínio face à resposta do utilizador.

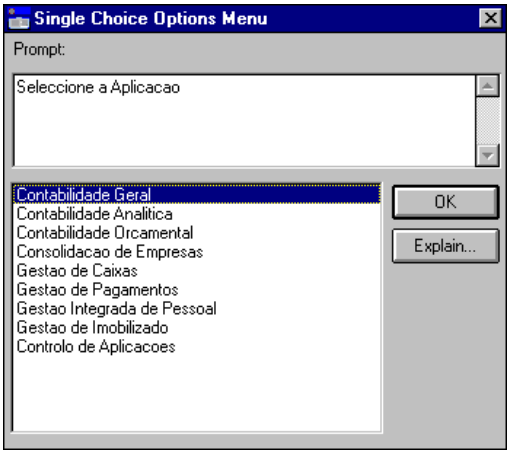

Figura 3 GISPSA interacção inicial

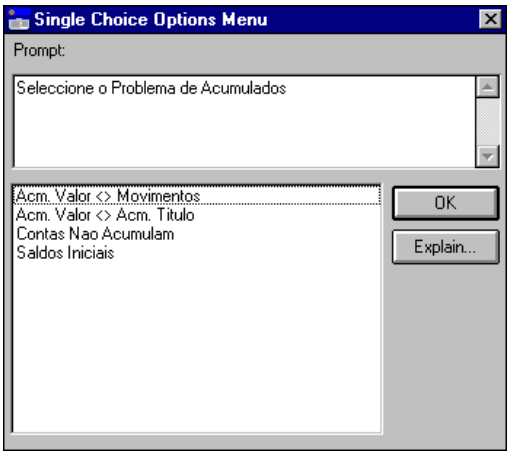

Figura 4 GISPSA exemplo de interacção pergunta-resposta

2.2.3 Limitações

Apesar de ser inovadora na área, esta aplicação foi, no início, bastante utilizada por alguns clientes tendo sofrido algum abandono de forma gradual ao longo do tempo de vida que já leva. Após uma análise, chegou-se à conclusão que as principais causas na origem desse abandono seriam as seguintes:

- Adicionar regras implicava o conhecimento da sintaxe das mesmas. Verificou-se que esta tarefa era demasiado complexa para um utilizador sem formação e sensibilidade suficientes na área da programação;
- O acto de proceder à edição de regras existentes na base de conhecimento, traduz-se numa tarefa que, para além de morosa, é essencialmente de risco no sentido em

que não existe uma verdadeira noção do impacto que estas alterações possam vir a ter no comportamento global do sistema;

- O sistema apenas pode ser instalado em mono-posto, o que obriga à obtenção de uma licença por cada instalação que se pretende realizar;
- O sistema não permite a integração com ambientes baseados em redes e sistemas distribuídos;
- Não existe nenhum mecanismo de verificação da base de conhecimento;
- O aspecto visual da aplicação está ultrapassado e é pouco apelativo.

2.3 *Web Expert Systems*

Actualmente, uma grande maioria dos sistemas periciais desenvolvidos são desenhados para um funcionamento em ambientes *web*. As razões apontadas para esta orientação são várias, destacando-se a evolução da própria tecnologia, as velocidades de transmissão de dados em redes, a possibilidade de acesso em múltiplas localizações geográficas (desde que assegurado o necessário acesso à rede) e ainda o desenvolvimento de novas ferramentas que possibilitam maior facilidade na criação destes novos sistemas.

Algumas das ferramentas que permitem actualmente o desenvolvimento de um sistema pericial baseado em ambientes *web* são as seguintes:

1. *WebFlex - LPA Flex*
2. *EXSYS CORVID*
3. *WebCLIPS*
4. *JCLIPS*
5. *JESS*
6. *Acquire*

WebFlex - Esta ferramenta permite a publicação de um sistema pericial desenvolvido em "*LPA Prolog*". Os sistemas periciais desenvolvidos a partir deste método, necessitam ainda de outra ferramenta para garantir o funcionamento, o "*LPA ProWeb Server*". É

possível adaptar um sistema pericial, já desenvolvido, para ser disponibilizado numa rede, sem ser necessário efectuar qualquer tipo de alteração. No entanto se for pretendido algum tipo de configuração extra, por exemplo incluir imagens ou alterar a aparência de alguns controlos, é uma alteração extremamente morosa e complexa.

*... WebFlex allows you to deliver your flex-based expert systems onto Intranets and Extranets easily. Just create your rules and frames and questions as per normal in flex, and WebFlex will construct a web server-based solution using ProWeb where all the questions are sent automatically configured for a standard HTML/Javascript browser...*Logic Programming Associaties Ltd (2010).

É possível alterar o HTML de saída que o *webflex* utiliza por defeito, no entanto esta configuração não é tão simples quanto poderá parecer à partida e obriga à alteração do sistema pericial.

Esta ferramenta funciona apenas em sistemas windows, e é necessária a obtenção de uma licença de utilização (não é gratuita).

EXSYS CORVID - É uma ferramenta composta por dois módulos: *Exsys CORVID Knowledge Automation Expert System Software* e *Exsys Servlet Runtime* EXSYS (2010).

Não foi possível encontrar nenhum sistema desenvolvido por esta ferramenta por motivos de confidencialidade e/ou vantagem competitiva. Estão, no entanto, disponíveis aplicações de demonstração na página oficial da empresa. Verificou-se que nenhuma das aplicações de demonstração disponíveis possui capacidades de dar explicações ou justificações.

De acordo com EXSYS, as aplicações construídas com esta ferramenta podem ser acedidas por *Servlet*, *Applet* ou *standalone* e esta ferramenta necessita da obtenção de uma licença de desenvolvimento.

WebCLIPS - Segundo Michael Giordano o *WebCLIPS* é uma implementação *CGI* que envolve o *C Language Integrated Production System* (CLIPS) como aplicação interna. O CLIPS é uma ferramenta de desenvolvimento de sistemas periciais.

Esta ferramenta foi desenvolvida em linguagem *C*, o que significa que em termos de portabilidade, desde que se possua o código fonte, este poderá ser compilado em qualquer sistema operativo, já que o compilador para a linguagem está disponível para todas as

versões existentes no mercado. No entanto, o *WebCLIPS* apenas foi desenvolvido para funcionamento em sistemas *windows* e *linux*.

O CLIPS é uma ferramenta livre que permite o desenvolvimento de sistemas baseados em regras. Pode ser integrado com algumas linguagens, das quais se destacam: *C*, *Java* (*JCLIPS*), *FORTRAN*, entre outras. Esta é, provavelmente, a ferramenta mais utilizada no desenvolvimento de sistemas periciais, devido à sua rapidez, eficiência e pelo facto de ser gratuita.

JCLIPS - É uma biblioteca que permite integração com *Java* e permite igualmente a utilização do motor do CLIPS a partir de uma classe *Java*. Esta interligação é feita através da importação do ficheiro (.dll ou .so consoante o sistema operativo) e permite a execução de uma instância, ou seja, permite ter apenas uma sessão de um sistema pericial de cada vez Maarten Menken (2010).

JESS - O *JESS* é uma extensão do CLIPS em *JAVA*. É basicamente uma biblioteca cujo objectivo não passa por gerar uma aplicação web mas sim servir de base para que possa ser integrada ou utilizada para esse efeito Ernest Friedman-Hill (2010).

Utiliza uma linguagem específica designada *JESS language*, equivalente ao CLIPS mas com novos componentes que a tornam uma linguagem complexa e de difícil utilização por alguém sem conhecimentos de inteligência artificial.

Acquire - É um produto bastante avançado do ponto de vista tecnológico face aos concorrentes, no sentido em que permite uma separação evidente entre a camada aplicacional e a camada de apresentação. Promove uma separação cliente-servidor e disponibiliza um *Software Development Kit* (SDK) que permite a integração com outras aplicações.

De acordo com Acquired Intelligence o produto está sujeito a uma licença de utilização e é disponibilizado ainda um serviço *online* de alojamento de sistemas periciais, o *myAcquire*, disponível mediante uma subscrição semestral.

2.3.1 Casos de estudo

Os sistemas periciais, mais especificamente aqueles que são baseados em sistemas *web*, têm conhecido nos últimos anos uma evolução gradual e consistente devido em grande parte à simplicidade e disponibilidade que possibilitam. Podem ser acedidos a partir de qualquer local no mundo, 24 horas por dia e sem qualquer tipo de limitação bastando

apenas que se garanta o acesso à rede onde o sistema pericial está colocado.

As interfaces são mais apelativas e chegam a fazer uso de imagens e outros meios gráficos ou de multimédia cujo objectivo não é apenas o de tornar os sistemas mais interessantes mas também mais eficazes, já que auxiliam de forma mais simples e esclarecida. Os controlos utilizados, as caixas de interacção e até mesmo as explicações e justificações (nos sistemas que o permitam) ajudam a ter uma ideia cada vez mais clara sobre o funcionamento dos sistemas, obtendo-se assim melhores resultados.

Alguns dos casos de estudo de sistemas periciais baseados em sistemas *web* existentes actualmente, têm como um dos seus maiores objectivos a disponibilidade global dos mesmos (são disponibilizados normalmente através da *internet*) e os mais conhecidos são:

- *Landfill Operation Management Advisor* (LOMA);
- *Medical Diagnosis Support System* (MDSS);
- *Whale watcher*.

2.3.1.1 *LOMA - Landfill Operation Management Advisor*

O *LOMA* é um sistema pericial *online* que analisa problemas operacionais, falhas e acidentes em aterros comuns. Além disso, o *LOMA* fornece conselhos sobre como um gestor pode evitar esses problemas, e em caso de ocorrência sobre como diminuir o impacto das suas consequências. Encontra-se disponível em `http://loma.civil.duth.gr/` e é de utilização pública.

O sistema pericial foi desenvolvido por Dokas recorrendo às ferramentas *WebFlex* e *LPA ProWeb Server*.

A arquitectura da aplicação *web* pode ser visualizada na figura 5 e consiste nos seguintes módulos Dokas (2005):

- Sistema pericial *LOMA*;
- Módulo de aconselhamento;
- Módulo operacional de problemas-causa;
- Eventos inicias;
- Problemas operacionais;

- Módulo de explicações;
- Módulo de submissão de conhecimento e informação.

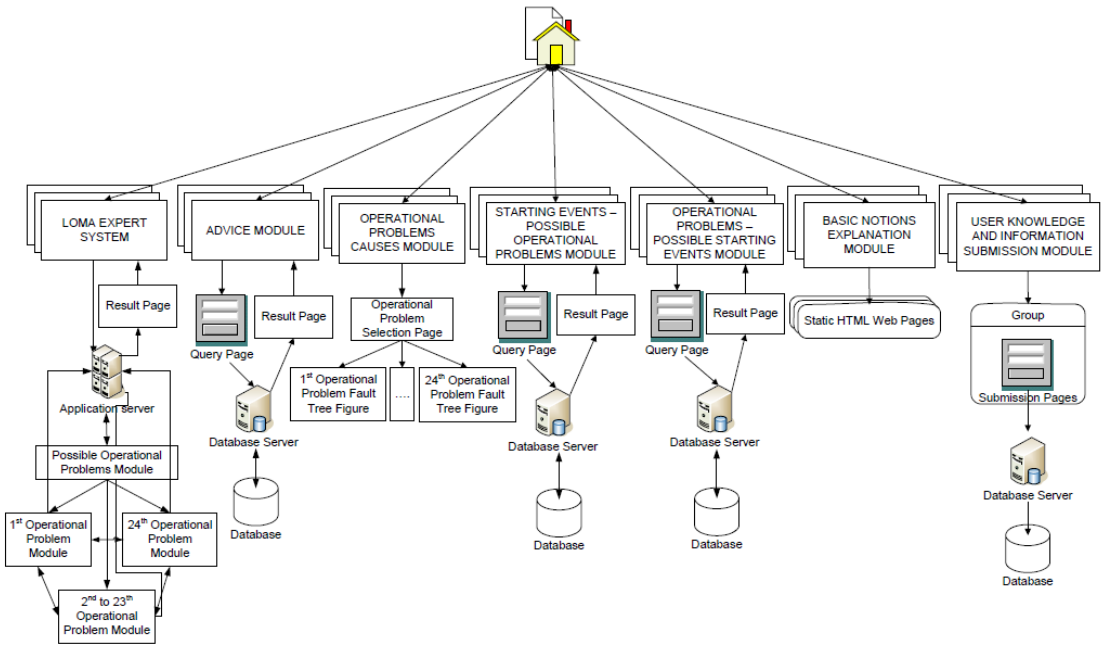

Figura 5 Arquitectura do LOMA

O módulo mais relevante para o âmbito desta dissertação é o "Sistema pericial *LOMA*" e é também o módulo mais importante para a aplicação *LOMA*.

É composto por vinte e quatro sub-módulos, cada um deles responsável por analisar um tipo específico de problema, e ainda por um sub-módulo extra que é responsável pelo controlo do sistema.

O sub-módulo extra questiona o utilizador que descreve o problema (ver 6 e 7) e com base na sua descrição apresenta uma lista de possíveis problemas (ver 8). Quando o utilizador selecciona um problema da lista de resultados é activado o respectivo sub-módulo que por sua vez irá fornecer informação de como prevenir o problema ou, no caso de este já ter acontecido, como minimizar as suas consequências (ver 9).

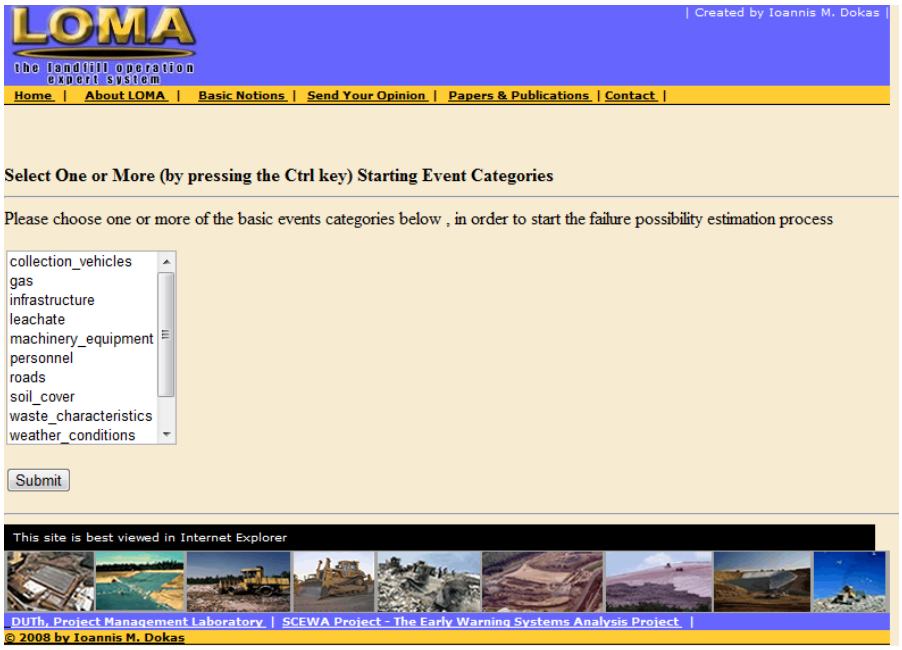

Figura 6 Página inicial do sistema pericial LOMA

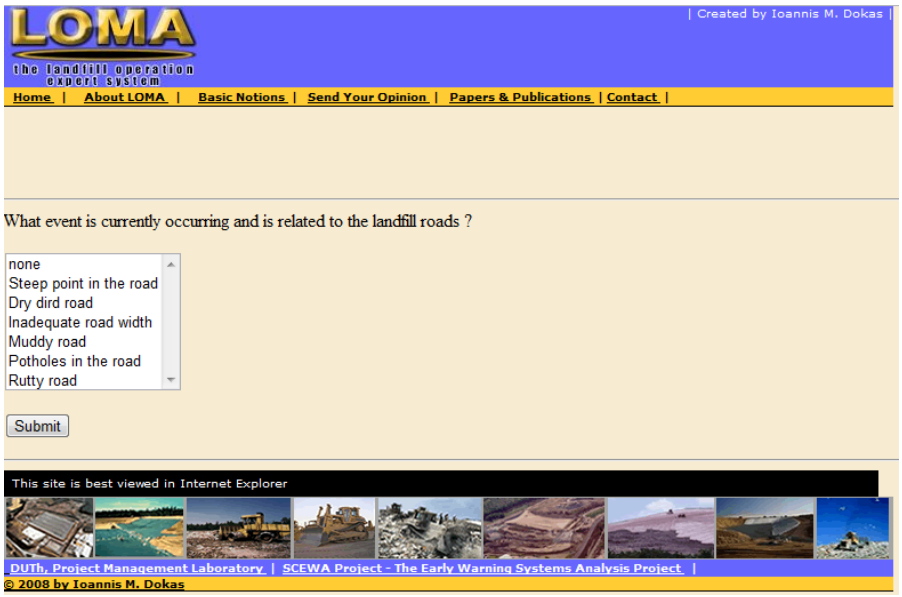

Figura 7 Sub-módulo extra

Figura 8 Sugestão de problemas do sub-módulo extra

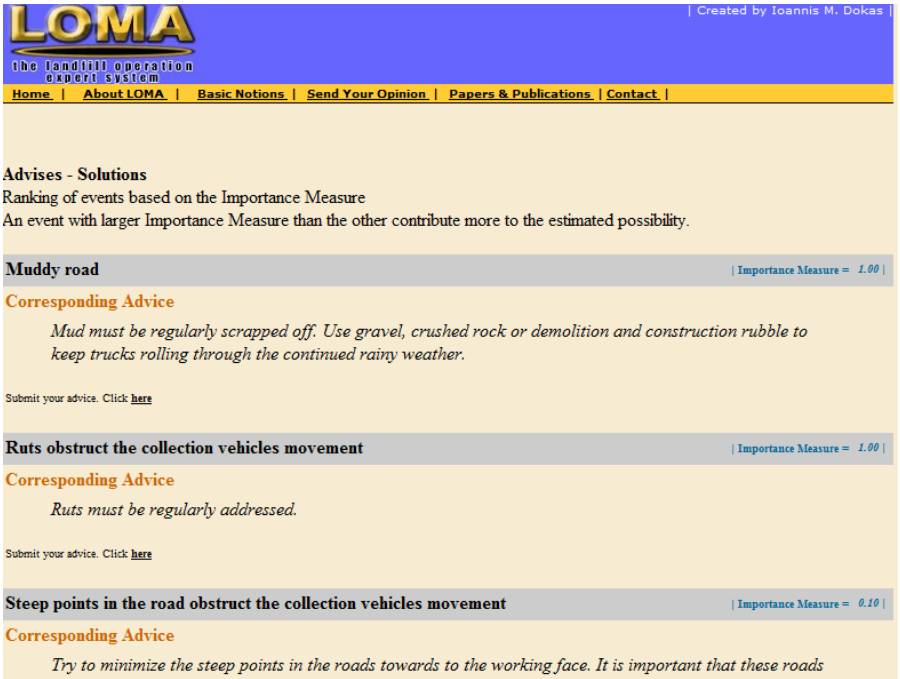

Figura 9 Sub-módulo especializado

2.3.1.2 *MDSS - Medical Diagnosis Support System*

O *MDSS* é um sistema pericial utilizado na área da medicina e foi desenvolvido para ajudar no diagnóstico e tratamento da diabetes. A diabetes pode afectar várias partes do corpo provocando diversos sintomas e complicações. Quando detectada atempadamente é mais facilmente controlável e menos perigosa. Por todos estes motivos foi criado um sistema pericial *online*, o *MDSS*, que permite inferir sobre uma série de sintomas para chegar a um diagnóstico.

Na opinião de Filho, uma das vantagens deste sistema pericial é que possibilita que médicos em ambientes rurais ou perigosos e de difícil acesso, com recursos limitados, possam aceder ao sistema pericial de forma a terem uma ajuda no diagnóstico desta doença. O que começou como um sistema pericial vocacionado e orientado para a diabetes, rapidamente cresceu e tornou-se algo mais do que isso, sendo hoje utilizado como sistema de diagnóstico para várias doenças.

No que diz respeito à sua arquitectura, o *MDSS* foi desenvolvido em *LPA Prolog* recorrendo aos módulos *ProWeb* e *ProData* e utilizando *Open Data Base Connectivity* (ODBC) para acesso à base de dados. Utiliza lógica *fuzzy* para raciocínio sobre incertezas para lidar com conhecimento parcial.

O sistema é composto por três módulos. No primeiro módulo, é possível um especialista definir sintomas, exames clínicos, exames de laboratório e exames relativos a uma ou mais doenças. É ainda possível a atribuição de valores (pesos estatísticos) ou de resultados que definem uma doença. Isso permite, por exemplo, que a pressão arterial possa estar associada ao diagnóstico de pressão alta, diabetes, gravidez de risco, ou insuficiência renal, entre outros. Informações de consultas anteriores e exames são automaticamente ligadas e analisadas sob fundamentação lógico temporal.

No segundo módulo, o sistema permite consultas através do qual os dados relevantes para o paciente são registados e relacionados com outros exames correlacionados. Durante a fase de diagnóstico, o sistema pode sugerir outros testes ou procedimentos para então inferir se há outros riscos e doenças a serem investigados.

No terceiro módulo é efectuado o diagnóstico. O sistema irá localizar automaticamente os diagnósticos já registados noutras consultas, incluindo aquelas feitas por outros médicos. Além disso, o sistema pesquisa níveis de doenças de acordo com os resultados dos exames. Pode também sugerir uma investigação mais aprofundada, sempre que os dados não sejam suficientes para garantir um diagnóstico preciso.

2.3.1.3 *Whale watcher*

O sistema pericial *Whale watcher*, é um sistema de demonstração da plataforma *Acquire*. Este e outros sistemas periciais similares estão disponíveis na página da *Acquire* (http://www.aiinc.ca/demos/).

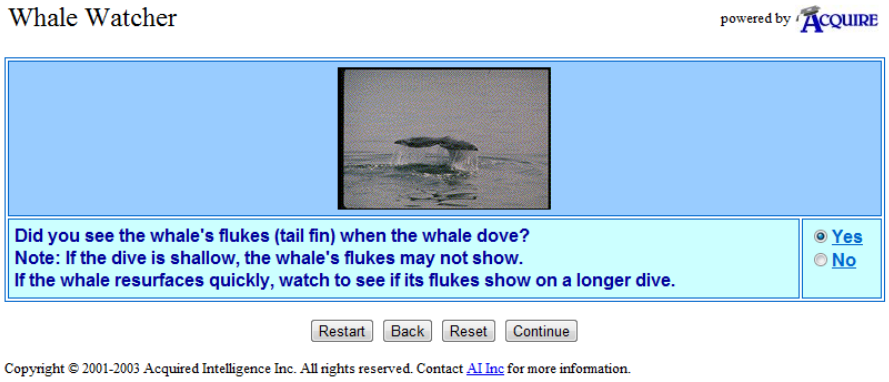

Figura 10 Página principal do sistema pericial Whale Watcher

Figura 11 Página final do sistema pericial Whale Watcher

Através das demonstrações facultadas pela *Acquire* não é possível concluir se existe algum tipo de sistema de explicações. No entanto, no final de cada sessão do sistema pericial é possível consultar um resumo (ver 11) que funciona como uma explicação muito básica da inferência realizada pelo sistema pericial, já que se limita apenas a mostrar os factos que foram respondidos durante a sessão.

3

Ferramentas utilizadas

Durante o desenvolvimento deste trabalho, foi necessário decidir sobre quais as ferramentas a utilizar para a criação do novo sistema pericial. Atendendo a algumas características próprias e objectivos traçados, era necessário utilizar alguma especificidade na escolha para garantir a exclusividade do resultado. Desenvolvimento *web*, capacidade de proceder a inferência sobre conhecimento representado através de regras, simplicidade de utilização, actualidade da tecnologia, entre outras características, influenciaram significativamente nesta escolha.

Entre os vários ambientes de desenvolvimento disponíveis, a opção recaiu sobre um que, para além de simples e eficiente, era utilizado regularmente em situação profissional pelo criador deste projecto pelo que aqui foi relativamente simples chegar a uma conclusão.

Indo de encontro às razões que levaram à escolha da linguagem do motor de inferência utilizada no projecto que deu origem a este trabalho, também aqui foi escolhida uma linguagem que possibilita o *backtracking* de forma eficaz e simplificada, característica essencial para o bom funcionamento do motor de inferência e que contribui largamente para o sucesso do sistema pericial.

3.1 Ambiente de desenvolvimento

O ambiente de desenvolvimento e respectiva linguagem de programação utilizados para a criação do interface do sistema pericial, foram seleccionados levando em conta alguns requisitos e factores considerados necessários e até mesmo essenciais para a solução final. A listagem dessas características é a seguinte:

- Baixo custo de disponibilização;

- Independência do sistema operativo;
- Simples escalabilidade;
- Visual apelativo;

Com base nesta lista de características, consideradas desejáveis ou até mesmo essenciais para a solução final, e considerando também a já longa experiência de utilização nesta plataforma, foi escolhida a linguagem *Java J2EE* utilizando a *framework Icefaces* e o portal *Glassfish*. Podemos considerar, para esta escolha, uma série de parâmetros para além dos já evidenciados:

- O *J2EE* é uma linguagem independente do sistema operativo;
- O *J2EE* caracteriza-se pela sua modularidade;
- *Glassfish* é um portal *open source* com suporte para *J2EE*;
- *Icefaces* é uma *framework JavaServer Faces (JSF)* compatível com o *Glassfish*.
- *Icefaces* permite a criação de interfaces gráficos de uma forma simples;
- Existência de boa documentação relativamente a todos os produtos.

Para além disto, podemos ainda considerar e referir outros produtos e componentes que, não sendo menos importantes, também foram utilizados durante o desenvolvimento deste trabalho. A lista completa de ferramentas é a seguinte:

- *Java J2EE* 6.0;
- *Eclipse Galileo* (com *plugins* para *Glassfish* e *svn*) - *Integrated Development Environment* (IDE) para *Java*;
- *Icefaces* 1.8.2;
- *GlassFish* - portal
- *Visual SVN Server* e *Tortoise* - controlo de versões
- *MikTeX* - *latex*
- *TexnicCenter* - IDE para *latex*

3.2 Motor de inferência

Em qualquer sistema pericial, o motor de inferência é o cerne do mesmo. O seu desenvolvimento obriga habitualmente a um esforço e concentração maiores porque aqui reside a forma como o sistema irá percorrer as várias etapas necessárias para alcançar uma solução.

Também neste trabalho o motor de inferência é um ponto essencial e a sua implementação teve um papel destacado no processo de análise realizado. Actualmente existem alguns produtos que podem ser utilizados, integral ou parcialmente, como motor de inferência de um sistema pericial, esses produtos podem ser *frameworks* ou *shells* ou ainda *Business Rule Management System (BRMS)*.

Os termos *framework* e *shell* são muitas vezes utilizados como sinónimo: uma *framework* é uma abstracção que permite agregar formas de resolução de problemas comuns; uma *shell* é similar no sentido em que disponibiliza uma camada de abstracção, uma "concha", que encapsula os serviços.

No sentido de tentar reutilizar um produto já existente, foi elaborado um estudo acerca dos actuais produtos e das suas características. A abordagem seguida para esse estudo foi uma abordagem por fases em que gradualmente se foi estreitando o conjunto de soluções aceitáveis mediante a utilização de alguns critérios.

Foram consideradas cinco fases:

- Fase I - Recolha de candidatos
- Fase II - Requisitos base
- Fase III - Viabilidade económica
- Fase IV - Viabilidade técnica
- Fase V - Prova de conceito

3.2.1 Fase I - Recolha de candidatos

A fase I teve como objectivo recolher alguns dos actuais produtos que possuam características desejadas para um sistema deste género. Não foram considerados alguns produtos que satisfaziam os requisitos mas que não permitem uma integração com Java. Foram identificados os seguintes produtos como potenciais integradores do projecto.

Produto	BRMS	Framework
Drools	X	X
Hammurapi Rules		X
ILog	X	
JCLIPS		X
JESS		X
JEOPS		X
JRuleEngine		X
OpenL Tablets	X	
OpenRules	X	
Prova language		X
Yprolog		X
Zilonis		X

Tabela 1 Listagem de produtos da Fase I

Drools - ou *JBoss Rules*, é um produto *open source* da RedHat (2010). Define uma linguagem específica para as regras. É desenvolvido em Java e utiliza um mecanismo de encadeamento directo baseado em regras, não implementa justificações.

Hammurapi Rules - é um produto desenvolvido em Java por Hammurapi Group. Este motor de inferência utiliza a sintaxe da linguagem Java para definição de regras. É possível definir grupos de regras em XML e está disponível um *plug-in* para *Eclipse* que permite a edição destes grupos. Suporta encadeamento directo e indirecto.

ILOG - *WebSphere ILOG Business Rule Management Systems* é, como o nome indica, um BRMS é um produto da IBM (2010) que requer licenciamento adequado. É necessária a instalação e configuração de alguns componentes de forma a que o produto possa ser utilizado, por exemplo: *Rule Team Server, Rule Execution Server, Http server*, entre outros.

JCLIPS - é um produto *open source* mantido por uma comunidade e permite a integração entre Java e CLIPS. O CLIPS "C Language Integrated Production System" é um sistema pericial desenvolvido em linguagem C Maarten Menken (2010).

JESS - é um produto comercial da *Sandia National Laboratories*, desenvolvido em Java. É necessário adquirir licenças para efectivar a utilização do mesmo Ernest Friedman-Hill (2010).

JEOPS - *Java Embedded Object Production System* (JEOPS) é uma *framework open source* desenvolvida num âmbito académico por Carlos Figueira Filho.

JRuleEngine - é um produto *open source* desenvolvido em Java e mantido por Mcarniel (2010). As regras são compiladas para classes Java.

OpenL Tablets - é um BRMS *open source* cuja principal vantagem é a integração com a folhas de cálculo Excel. Permite a utilização de tabelas de decisão no formato Excel OpenL (2010).

OpenRules - é um produto da OpenRules, Inc. sujeito a licença de utilização. É totalmente orientado para a folha de cálculo Excel da Microsoft.

Prova language - é um produto *open source* desenvolvido em Java por Kozlenkov, Alex and Paschke. Disponibiliza uma linguagem muito semelhante ao prolog. É um produto bastante direccionado para a comunicação entre agentes, expondo inclusivé um sistema multi-agente.

Yprolog - é um produto *open source* desenvolvido em Java por Boris van Schooten. É uma réplica do prolog desenvolvida totalmente em Java. Foi evoluindo ao longo do tempo a partir de produtos como o WProlog e o XProlog.

Zilonis - é um produto desenvolvido em Java por Elie Levy. O formato das regras é semelhante ao CLIPS.

3.2.2 Fase II - Requisitos base

A fase II teve como objectivo filtrar a lista de candidatos no sentido de excluir produtos que não se enquadrem directamente com os requisitos do sistema a desenvolver. Assim sendo, foram excluídos alguns dos candidatos já indicados e os principais motivos da exclusão são:

- Não permitirem a integração com um portal;
- Formato das regras é fechado não possibilitando possíveis e futuras evoluções;
- Não disponibilização de acesso aos serviços internos, mais concretamente não fornecem uma *framework* de controlo. Desta forma não é possível reutilizar partes do motor de inferência ou do sistema de gestão de regras sem que este seja feito através do próprio produto.

Candidatos excluídos pelas razões atrás expostas:

Produto
ILog
OpenRules
OpenL Tablets

Tabela 2 Listagem de produtos excluídos na Fase II

3.2.3 Fase III - Viabilidade económica

Sendo um parâmetro com importância relativa dada a possível aplicação comercial de um produto baseado neste trabalho, não deixa de ser importante para o estudo realizado. Desta forma, a fase III teve como objectivo filtrar a lista de resultados no sentido de excluir produtos que não sejam economicamente viáveis, quer por motivos de licenciamento comercial quer por motivos de licenças de utilização.

Foram então considerados os produtos constantes da tabela 3 como não viáveis, no sentido em que existem produtos *open source* similares para o efeito pretendido.

Produto
Drools
JESS

Tabela 3 Listagem de produtos excluídos na Fase III

3.2.4 Fase IV - Viabilidade técnica

Esta fase teve por objectivo filtrar a lista de resultados de forma a que os produtos não viáveis do ponto de vista técnico não sejam considerados para a escolha final.

Os produtos que não foram considerados viáveis por questões técnicas são:

Produto	**Razão**
JEOPS	Documentação disponível bastante vaga, projecto descontinuado e não estável
JRuleEngine	As regras são compiladas para Java, não permite escalabilidade sem compilar o projecto
Zilonis	Não está preparado para ser utilizado como *framework*, apenas permite uma aplicação *stand-alone*
Hammurapi Rules	As regras são compiladas para Java, não permite escalabilidade sem compilar o projecto

Tabela 4 Listagem de produtos excluídos na Fase IV

3.2.5 Fase V - Prova de conceito

Nesta última fase restaram três produtos: *JClips*, *Prova* e *YProlog*. Após a realização de uma prova de conceito com estes produtos, foram encontradas algumas vantagens e desvantagens. Esta análise pode ser consultada nas tabelas 5, 6 e 7.

JClips	
Vantagens	**Desvantagens**
• Linguagem CLIPS .	• Apenas permite 1 execução, de forma a permitir várias utilizações do motor de inferência é necessário implementar lógica específica; • Ausência de documentação.

Tabela 5 JClips - Vantagens e desvantagens

Prova	
Vantagens	**Desvantagens**
• Linguagem Prolog; • Permite múltiplas execuções em simultâneo; • Disponibiliza exemplos de utilização; • Última actualização em 10 de Junho de 2010.	• Instável; • Bastante ocupação de recursos (CPU e memória); • Disponibilização de pouca documentação; • Ao invocar um predicado, não instancia apenas as variáveis na primeira vez mas devolve uma lista de todos os resultados possíveis, similar ao predicado *findall* do *LPA Prolog*.

Tabela 6 Prova - Vantagens e desvantagens

YProlog	
Vantagens	**Desvantagens**
• Linguagem Prolog; • Permite múltiplas execuções em simultâneo; • Disponibiliza documentação e exemplos de utilização; • Operações básicas do Prolog têm o mesmo comportamento.	• Última actualização em 21 de Abril de 2006.

Tabela 7 YProlog - Vantagens e desvantagens

3.2.6 Conclusão e fundamentação da escolha

Tratando-se do núcleo do sistema pericial e sabendo que não existe actualmente um motor de inferência que corresponda totalmente aos requisitos necessários, foi decidido

desenvolver um motor de inferência de raiz recorrendo a um produto que permitisse um desenvolvimento rápido e com uma curva de aprendizagem pouco acentuada. Para tal e depois do estudo realizado perante as várias hipóteses disponíveis, foi decidido optar pelo *YProlog*.

Na última fase do estudo apresentado, entre os candidatos disponíveis, este aparece como o mais viável, quer pelos aspectos de adaptabilidade, funcionamento e facilidade de utilização, quer ainda pelo número reduzido de desvantagens encontradas face aos restantes.

4

Wexsys

Este trabalho baseia-se na criação de um sistema pericial baseado no GISPSA e o objectivo principal passa pela integração do novo sistema num ambiente de acesso global ou mais especificamente, num portal web. Para atingir tal meta, foi descrito nos capítulos anteriores quais as ferramentas a utilizar sendo que agora vamos compreender como é que a solução em si foi desenvolvida.

O sistema que nasceu a partir desta implementação foi baptizado com o nome wexsys - "Web Expert System". Partindo da base do GISPSA, o wexsys possui um motor de inferência totalmente novo, criado especificamente no âmbito deste trabalho e ainda uma base de conhecimento. Foi decidido adoptar o princípio de utilização de regras para a representação da base de conhecimento.

De referir ainda que o raciocínio utilizado no motor de inferência segue uma filosofia do tipo *backward chaining*, princípio já descrito nos capítulos anteriores.

Detentor de uma interface considerada actual para os parâmetros utilizados nos dias que correm, este novo sistema pericial foi criado através da utilização de *J2EE* e *Icefaces* que lhe conferem um aspecto renovado e extremamente apelativo aos utilizadores. São utilizadas técnicas tais como *Ajax*, que providenciam uma dinâmica de funcionamento muito interessante do ponto de vista da utilização.

4.1 Arquitectura global

A solução foi projectada de acordo com a arquitectura *J2EE*, ou seja, utilizando uma separação em módulos ou componentes. Foram criados dois módulos principais: *pWexsys.EAR* e *Wexsys.EAR*, estes módulos são a separação da aplicação em duas grandes camadas, a camada *web* e a camada de negócio.

Esta separação permite a instalação da aplicação num único servidor, ou uma instala-

ção distribuída por vários servidores possibilitando também a utilização de *clusters*. Esta separação torna-se bastante útil se pretendermos expandir a aplicação para uma utilização mais intensa ao nível dos acessos. A possibilidade de instalação em múltiplas máquinas permite a obtenção de melhores resultados quando confrontada com a utilização de um ambiente com apenas um único servidor.

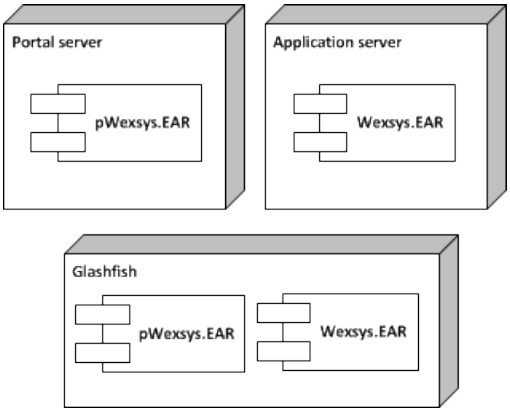

Figura 12 Arquitectura da solução

4.1.1 Módulo pWexsys.EAR

Este é o módulo responsável pela componente visual da solução, também conhecida por *Presentation Layer*. Este módulo é composto pelos componentes descritos na figura 13.

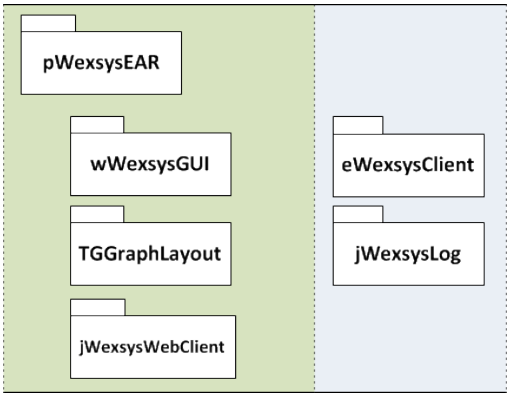

Figura 13 Arquitectura do módulo visual

- **pWexsysEAR** - responsável pela criação da aplicação e da declaração das dependências externas e internas;

- **wWexsysGUI** - componente principal que contém as *JSP*'s, *servlet* e *managed beans*;
- **jWexsysWebClient** - classes utilizadas na camada visual;
- **TGGraphLayout** - componente externo que disponibiliza um controlo que permite a definição de um grafo, este componente vai ser descrito na secção 4.1.3.3;
- **jWexsysLog** - permite o registo da ocorrência de erros;
- **eWexsysClient** - classes utilizadas como objectos de transferência, bem como um interface para o módulo de negócio.

4.1.2 Módulo Wexsys.EAR

Este é o módulo responsável pela componente de negócio da aplicação, também conhecida por *Business Layer*, é o centro de controlo de toda a solução. Este módulo é composto por alguns componentes, como podemos ver na figura 14.

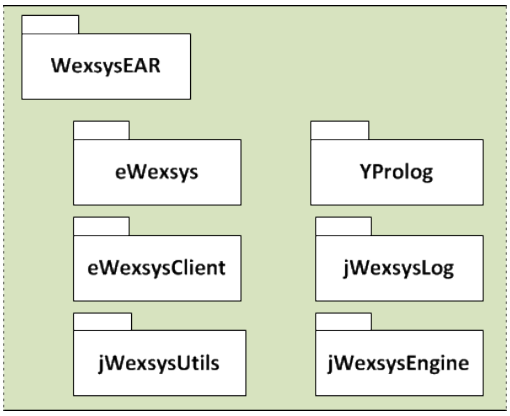

Figura 14 Arquitectura do módulo de negócio

- **WexsysEAR** - responsável pela criação da aplicação e da declaração das dependências externas e internas.
- **eWexsys** - componente principal responsável pela criação dos serviços:
- **eWexsysClient** - classes utilizadas como objectos de transferência, bem como um interface para o módulo de negócio;
- **jWexsysUtils** - classes utilitárias internas;

- **jWexsysLog** - componente auxiliar para permitir o tratamento de erros;
- **jWexsysEngine** - responsável pela implementação do motor de inferência;
- **YProlog** - permite a interpretação de código prolog. Este componente vai ser descrito na secção 4.1.3.3.

4.1.3 Interligação e integração dos módulos

Como já foi referido, o *wexsys* foi desenvolvido respeitando a arquitectura J2EE, como tal os componentes que integram a aplicação foram também criados seguindo esse conceito. Todos os componentes foram estruturados de forma a poderem ser reutilizados com o mínimo de dependências e toda a aplicação foi projectada de forma a retirar o máximo partido desta divisão.

Podemos ver na figura 15 um diagrama que permite verificar de que forma os componentes estão a ser utilizados. O diagrama tem três agrupamentos que pretendem descrever, utilizando uma linguagem *Object Oriented* (OO), a sua visibilidade. Os temos *public*, *protected* e *private* são utilizados como analogias aos respectivos conceitos na linguagem OO.

4.1.3.1 Camada de acesso público

A camada de acesso público expõe os métodos de interacção com a aplicação, sendo composta por três componentes: wWexsysGUI, jWexsysWebClient e eWexsysClient.

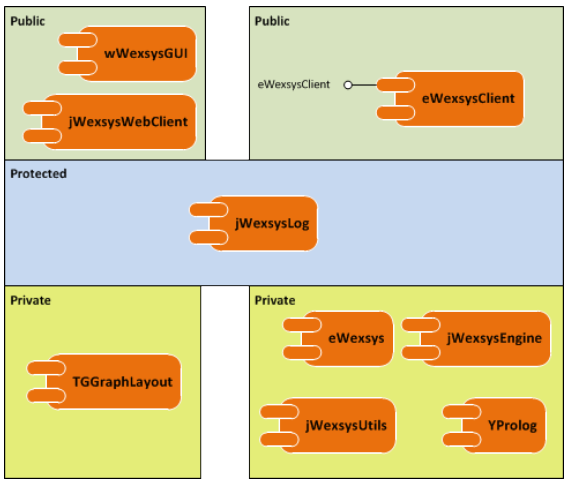

Figura 15 Wexsys - Integração dos componentes

Os componentes eWexsysClient e jWexsysWebClient, contêm interfaces de ligação entre a camada visual e a camada de negócio.

eWexsysClient - disponibiliza as classes designadas *Data Transfer Object* (DTO) e ainda o interface para os serviços disponibilizados pelo wexsys.

jWexsysWebClient - disponibiliza classes que caracterizam os conceitos de negócio mas que contêm atributos adicionais. A razão da existência deste componente justifica–se com a eventual expansão da aplicação para uma nova camada visual. Os exemplos de código 1 e 2 permitem-nos observar uma utilização de atributos extra na classe visual para controlo de ocorrência de erros.

```
public class RuleView {

  private int level = 0;
  /**
   * Rule identifier
   */
  private String identifier = null;

  private String identifierErr = null;

  private boolean identifierInErr = false;

  /**
   * Rule description
   */
  private String description = null;

  private String descriptionErr = null;

  private boolean descriptionInErr = false;
  ...
}
```

Código 1 Excerto da classe RuleView

```
public class Rule implements Serializable {
```

```
2
    private int level = 0;
4   /**
     * Rule identifier
6    */
    private String identifier = null;
8
    /**
10   * Rule description
     */
12  private String description = null;
    ...
14 }
```

Código 2 Excerto da classe Rule

O outro componente envolvido na camada pública, o *wWexsysGUI*, é o responsável pela interacção com aplicação. É o componente que disponibiliza as páginas e o responsável pela coordenação de todo o processo *web*.

4.1.3.2 Camada de acesso protegido

A camada de acesso protegido disponibiliza componentes que podem ser utilizados pelos dois módulos, mas que não estão disponíveis para o exterior. Neste caso existe apenas um componente nestas condições, o *jWexsysLog* cuja função é assegurar o registo de erros.

jWexsysLog - componente responsável pelo registo de erros, utiliza internamente a *framework log4j*.

4.1.3.3 Camada de acesso privado

A camada de acesso privado incluí todos os componentes responsáveis por processamento interno. Os componentes responsáveis pelo processamento interno são: *eWexsys*, *jWexsysEngine* e *jWexsysUtils*. Também incluídos no conceito de processamento interno estão os componentes *TGGraphLayout* e *YProlog*.

eWexsys - é o componente responsável pela implementação do interface de negócio, delegando as funcionalidades em classes especializadas ou no componente responsável pela integração com o núcleo da aplicação, o *jWexsysEngine*.

jWexsysEngine - é o componente responsável pelo núcleo da aplicação, ou seja, é o responsável pela integração do java com o prolog e vice-versa. É o componente responsável pelo mapeamento das classes de java para as classes de comunicação com o prolog.

jWexsysUtils - é um componente auxiliar que disponibiliza e encapsula utilitários para uma melhor organização interna da aplicação.

TGGraphLayout - é um componente também conhecido por *TouchGraph*, é um projecto mantido por uma comunidade e está disponível no *sourceforge* Alexander Shapiro (2010)

YProlog - este componente foi o escolhido durante a fase de estudo da implementação para ser a base do motor de inferência (3.2.1).

4.1.3.4 Fluxo de invocação

A aplicação foi desenvolvida de forma a que todos os pedidos seguissem um mesmo fluxo de invocação de forma a uniformizar o processo. Na figura 16 temos uma relação das classes principais da aplicação.

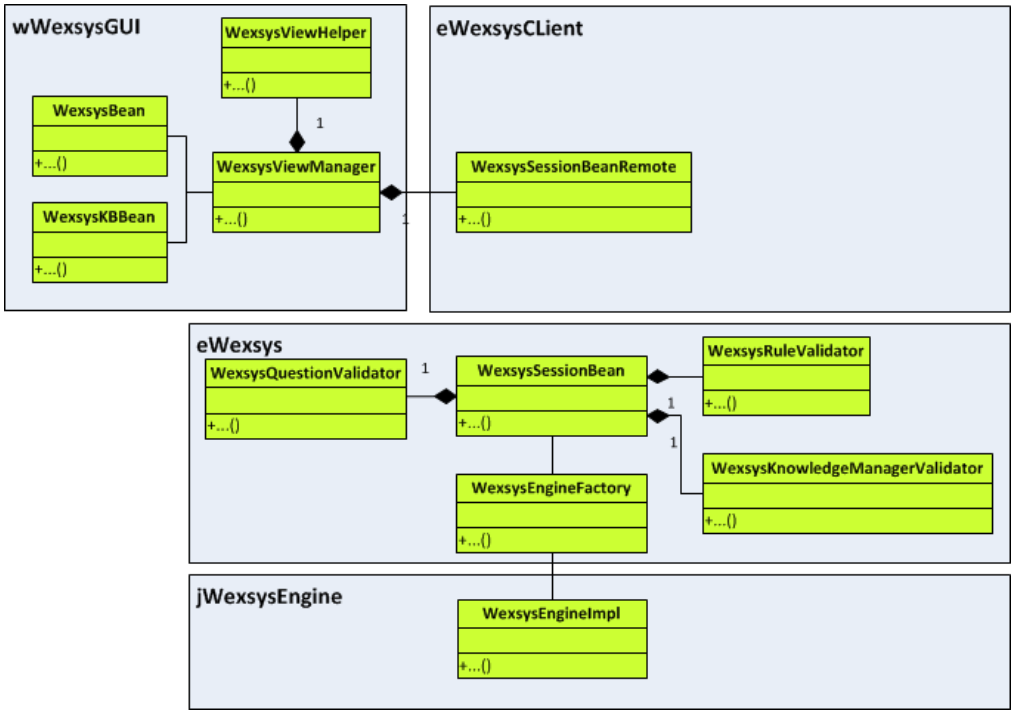

Figura 16 Diagrama de classes principais

O fluxo de invocação inicia-se através do WexsysBean no caso de funcionalidades

do motor de inferência, ou através do WexsysKBBean no caso de funcionalidades da base de conhecimento. Ambos os *managed beans* delegam no *WexsysViewManager* que recorrendo à classe *WexsysViewHelper* converte as classes *view* para classes DTO. A classe *WexsysViewManager* por sua vez delega no *session bean WexsysSessionBean* que mediante o tipo de pedido delega nas classes de validação ou nas classes do motor de inferência *WexsysEngineImpl*.

Este processo pode ser observado na figura 17 que representa um diagrama de sequência exemplificativo do fluxo de invocação genérico.

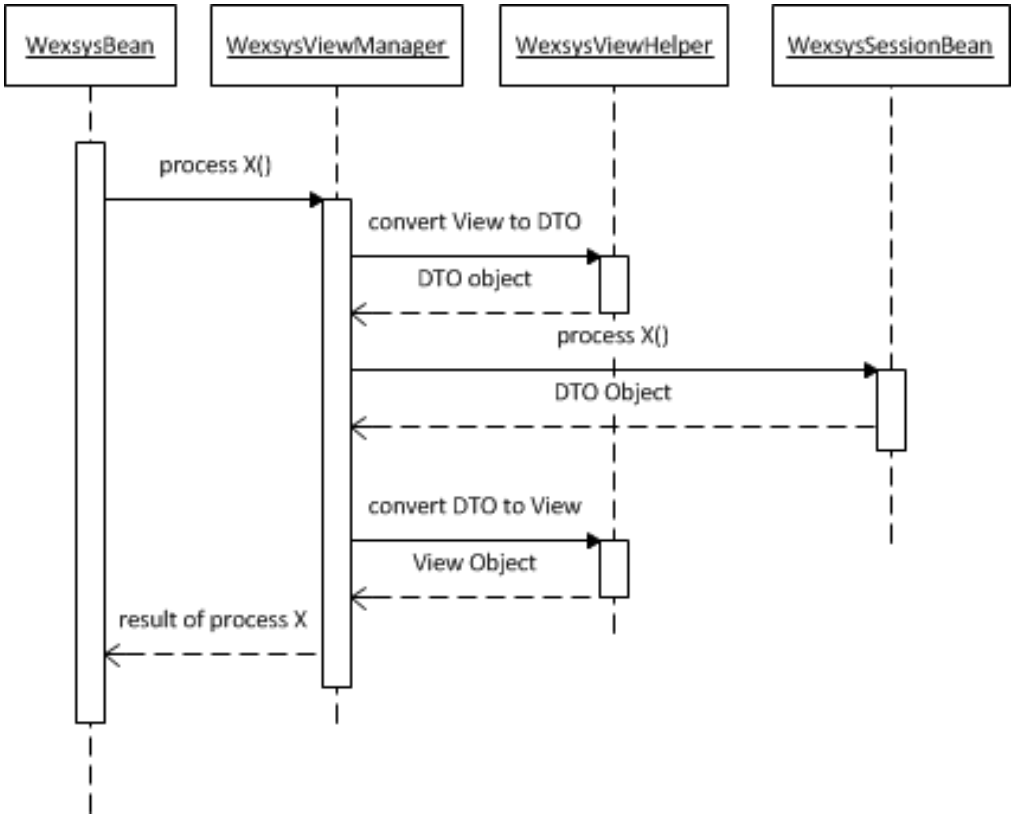

Figura 17 Diagrama de sequência do fluxo de invocação

4.1.4 Portal

O wexsys tem como ponto de entrada na aplicação uma página inicial que permite a identificação do utilizador no sistema pericial. Como se trata de um trabalho académico a parte de validação e autenticação de utilizadores não foi contemplada, como tal é possível autenticar-se com qualquer utilizador. Para além da autenticação é ainda possível indicar para a sessão actual qual o mínimo grau de confiança para a apresentação de uma resposta.

O endereço da aplicação respeita o seguinte formato `http://servidor:porta/wWexsysGUI/`, onde servidor e porta são os dados de configuração do *Portal Server*.

Figura 18 Wexsys - Página inicial

Quando o utilizador inicia a utilização do sistema, ficam disponíveis três funcionalidades que serão referidas nas secções seguintes.

4.1.4.1 *Templates*

Um dos aspectos comuns a toda a aplicação é o aspecto gráfico, tendo este sido implementado recorrendo a *Facelets*. Um template *facelets* define-se pela criação de uma página modelo em que existem zonas dinâmicas. Este conceito é conhecido de outras tecnologias, por exemplo *Web Parts* ou *Master Pages* desenvolvidos pela *Microsoft*.

O código 3 permite ver como foi desenvolvido o template principal.

```
<html
xmlns="http://www.w3.org/1999/xhtml"
xmlns:ui="http://java.sun.com/jsf/facelets"
xmlns:f="http://java.sun.com/jsf/core"
xmlns:ice="http://www.icesoft.com/icefaces/component">

<ice:outputDeclaration
doctypeRoot="HTML"
doctypePublic="-//W3C//DTD XHTML 1.0 Strict//EN"
doctypeSystem="http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.
 dtd" />
```

```
<head>
 <title>wexsys - #{pageTitle}</title>
  [meta tags e link to css files]
</head>
<body>
<f:view locale="#{facesContext.externalContext.requestLocale}">
 <ice:loadBundle basename="com.wexsys.gui.resources.bundleRes"
  var="bundleRes" />
  <ice:form partialSubmit="true">
   <div id="wrapper">
    <div id="logo">
     <h1><a href="#">WExSys </a></h1>
     <h2>web expert system</h2>
    </div>
    <div id="header">
     <div id="menu">
      [Menu content]
     </div>
    </div>
   </div>
   <div id="page">
    <div id="content">
     <ui:insert name="content">
      Default Content
     </ui:insert>
    </div>
    <div id="sidebar">
     <ui:insert name="sidebar">
      Sidebar content
     </ui:insert>
    </div>
    <div style="clear: both;">
    </div>
   </div>
   <div id="footer">
    Footer content
   </div>
  </ice:form>
```

```
  </f:view>
</body>
</html>
```

Código 3 Código do template principal do portal

Na listagem de código 3 podemos ver a vermelho as áreas que foram definidas como dinâmicas, no código 4 também a vermelho podemos ver as respectivas áreas num cenário de implementação de uma página.

```
<ui:composition
template="WEB-INF/inc-templates/main-template.jspx"
xmlns="http://www.w3.org/1999/xhtml"
xmlns:ui="http://java.sun.com/jsf/facelets"
xmlns:f="http://java.sun.com/jsf/core"
xmlns:ice="http://www.icesoft.com/icefaces/component">


  <ui:define name="content">
    [...]
  </ui:define>

  <ui:define name="sidebar">
    [...]
  </ui:define>

</ui:composition>
```

Código 4 Código da página inicial do portal

4.1.5 Suporte multi-língua

O *wexsys* foi desenvolvido considerando um factor bastante relevante numa aplicação actual, o suporte multi-língua. Este suporte é uma das vantagens do sistema e foi implementado recorrendo a ficheiros de recursos (.properties) de forma a exteriorizar todos os textos de saída.

Desta forma para dar suporte a uma nova língua apenas teremos que traduzir os textos nos ficheiros de recursos e de alterar a aplicação de forma a permitir seleccionar o idioma.

A alteração a ser efectuada na aplicação é muito simples já que basta passar o valor do atributo "*locale*" da *tag* "*f:view*" como sendo a língua pretendida.

4.2 Motor de inferência

Para este trabalho, foi definida a criação de um motor de inferência de raiz que pudesse efectivamente substituir o utilizado no sistema base. O novo motor desenvolvido terá que suportar uma série de especificações, a começar pela capacidade de operar num ambiente multi-utilizador.

O motor de inferência deve estar preparado para lidar com conhecimento incerto, foi escolhido a técnica dos factores de certeza por ser uma técnica simples de reproduzir e eficaz na sua utilização.

Para além disto e tratando-se um sistema pericial, o motor de inferência deverá ser independente da base de conhecimento utilizada e deverá, idealmente, suportar um sistema de explicações que responda a eventuais pedidos por parte dos utilizadores que pretendam ver mais detalhadamente quais as regras que conduziram à conclusão obtida. Também será possível saber porque é que determinada questão é colocada (explicações vulgarmente denominadas de como e porquê).

4.2.1 Multi-utilizador

Quando se pretende que um sistema pericial tenha uma utilização mono-posto (como era o caso do GISPSA), apenas é criada uma instância por cada vez que é chamado o motor de inferência. No nosso caso, tendo sido estabelecido que o novo sistema teria que permitir uma utilização multi-utilizador, é necessário desenvolver os mecanismos necessários para permitir execuções de várias instâncias para o novo motor de inferência.

A classe *WexsysEngineFactory5* é a responsável por criar uma instância do motor de inferência por cada utilizador que inicia a utilização do sistema. Esta classe tem um método do tipo *static*, o método *getInferenceEngine*, que tem como parâmetros de entrada um utilizador e devolve uma instância do motor de inferência. As instâncias activas são guardadas internamente e sempre que é invocado o método, é obtida a respectiva instância. No caso de ainda não existir uma instância, então é criada uma nova e guardada.

```
1 /**
    * Creates an instance of InferenceEngine
```

```
3     *
      * @param usernameKey
5     *              The usernameKey to use
      *
7     * @return InferenceEngine instance
      * @throws WexsysArgumentException
9     */
    public static WexsysEngineImpl getInferenceEngine(String
     usernameKey)
11       throws WexsysArgumentException {
      if (getRuleUrl() == null || getRuleUrl().length() == 0) {
13      throw new WexsysArgumentException("mainKnowledgeBaseUrl,␣
         ruleUrl");
      }
15    WexsysEngineImpl inferenceEngine = null;

17    /** Have we already created an instance */
      if (getInferenceEngines().containsKey(usernameKey)) {
19      /** Obtain the instance associated to the specified key
         */
        inferenceEngine = getInferenceEngines().get(usernameKey);
21    } else {
        /** Create a new instance and store it in the holder */
23      inferenceEngine = new WexsysEngineImpl(usernameKey,
         ruleUrl, kbsUrl);
        getInferenceEngines().put(usernameKey, inferenceEngine);
25    }

27    return inferenceEngine;
    }
```

Código 5 Método responsável pela utilização multi-utilizador em paralelo

4.2.2 Base de conhecimento inicial

Ao partir para o desenvolvimento de um sistema pericial, é necessário adoptar uma base de conhecimento inicial simples, que sirva de suporte ao desenvolvimento do mesmo.

A base de conhecimento inicial foi adoptada a partir do estudo de Merritt (1989). Foi seleccionada devido a duas características principais:

- simplicidade - no sentido em que as regras são pequenas e simples de interligar;
- abrangência - existem alguns cenários possíveis de serem testados com o protótipo, como por exemplo raciocínio sobre incerteza e conclusões intermédias.

A base de conhecimento inicial escolhida tem a seguinte estrutura:

- **rule** 1 **if** not turn_over and battery_bad **then** problem is battery cf 100.
- **rule** 2 **if** lights_weak **then** battery_bad cf 50.
- **rule** 3 **if** radio_weak **then** battery_bad cf 50.
- **rule** 4 **if** turn_over and smell_gas **then** problem is flooded cf 80.
- **rule** 5 **if** turn_over and gas_gauge is empty **then** problem is out_of_gas cf 90.
- **rule** 6 **if** turn_over and gas_gauge is low **then** problem is out_of_gas cf 30.

4.2.3 Protótipo do motor de inferência

Para o desenvolvimento do protótipo inicial do motor de inferência, foi necessário definir alguns aspectos essenciais, tais como:

1. Formato da regras;
2. Formato das questões;
3. Interacção assíncrona de questões e respostas;
4. Algoritmos a utilizar.

4.2.3.1 Formato das regras

O formato das regras foi definido tendo em conta alguns pressupostos. Entre eles, salientam-se a necessidade da utilização de graus de confiança e a capacidade de guardar uma descrição para cada regra.

O formato escolhido para a representação de uma regra foi o seguinte:

```
rule(
  <codigo>, 'Descrição da regra',
  lhs([
       ev(<premissa>, <valor da premissa>), ...,
       ev(<premissa_n>, <valor da premissa_n>)
      ]),
  rhs(ev(<solução>, <valor da solução>), CF)
).
```

Na definição do <LHS> é possível definir uma lista de premissas e respectivos valores no caso da regra conter mais do que uma premissa. Um aspecto relevante relacionado com a lista de premissas é que esta considera apenas o operador conjunção entre premissas.

4.2.3.2 Formato das questões

O formato das questões foi também definido assumindo alguns princípios tais como a possibilidade de respostas múltiplas e ainda a atribuição de um texto quer à questão a ser apresentada quer às hipóteses colocadas.

O formato escolhido para a representação de uma questão foi o seguinte:

```
question(
  turn_over,
  'O carro liga?',
  [
    answer(<valor>, <descrição do valor>), ...,
    answer(<valor_n>, <descrição do valor_n>)
  ],
  <escolha_multipla = 1 / 0>
).
```

4.2.3.3 Interacção assíncrona de questões e respostas

Numa qualquer execução de um *software* instalado e executado localmente numa máquina, o funcionamento normal será a execução de um conjunto de instruções até ao ponto em que seja necessária a interacção com um utilizador. Quando isso acontece, o programa fica suspenso até que o utilizador responda ao que lhe foi solicitado, continuando depois com a sua execução a partir do ponto onde parou.

No caso de um programa desenvolvido em *prolog* e mais especificamente um sistema pericial, o conceito atrás descrito é exactamente o mesmo já que o programa apenas suspende a execução quando é necessário colocar uma questão que o utilizador terá que forçosamente responder para dar continuidade ao programa.

Se, no entanto, considerarmos a realidade de uma aplicação *web*, este princípio já não será tão linear porque um servidor *web* recebe um pedido, executa-o e devolve um resultado. O mesmo será dizer que, numa execução de um componente *prolog* a partir de um servidor *web*, a pilha de execução é interrompida quando é necessário questionar o utilizador mas não é possível continuar a execução da mesma a partir do ponto onde parou. Em vez disso, será necessário desfazer a pilha de execução criada e, assim que existir a resposta do utilizador (é feito novo pedido ao servidor), a execução é realizada novamente desde o início mas desta vez considerando a resposta recebida.

As razões para este comportamento justificam-se pela utilização do protocolo HTTP em que é necessário existir um pedido e uma resposta. O que obriga a que a colocação da pergunta e a resposta sejam assíncronos, podendo existir entre as duas acções vários pedidos (pedidos de explicações por exemplo).

Este "problema" levou à necessidade do desenvolvimento de um mecanismo que permita um funcionamento coerente num programa que utilize (como neste caso) o *prolog*, simulando o comportamento que este teria numa situação de execução local.

O mecanismo proposto foi a alteração do predicado responsável pelo processo recursivo, especificando um atributo auxiliar de controlo que acompanha a execução do predicado. Esse atributo auxiliar, quando detectada a necessidade de colocação de uma questão, é instanciado e impede a prossecução do predicado, retornando a execução para a *web* com a respectiva questão.

Este mecanismo permite uma comunicação assíncrona com a parte *web* mantendo as vantagens da recursividade do *prolog*.

4.2.3.4 Estrutura da base de conhecimento

A base de conhecimento, após a definição do formato de representação das regras e das questões, é a seguinte:

```
question(turn_over, 'O carro liga?',
    [answer(yes, 'Sim'), answer(no, 'Nao')],0).

question(lights_weak, 'As luzes estao fracas?',
```

```
    [ answer(yes, 'Sim'), answer(no, 'Nao')],0).

question(radio_weak, 'O radio esta fraco?',
    [ answer(yes, 'Sim'), answer(no, 'Nao')],0).

question(smell_gas, 'Cheira a gasolina?',
    [ answer(yes, 'Sim'), answer(no, 'Nao')],0).

question(gas_gauge, 'O que marca o ponteiro do deposito da
 gasolina?',
    [ answer(empty, 'Vazio'), answer(high, 'Suficiente'), answer
     (low, 'Reserva')],0).

rule(rule_1, 'Rule 1',
   lhs([ev(turn_over, no), ev(battery_bad, yes)]), rhs(ev(
    problem, battery), 100)).

rule(rule_2, 'Rule 2',
   lhs([ev(lights_weak, yes)]), rhs(ev(battery_bad, yes), 50)).

rule(rule_3, 'Rule 3',
   lhs([ev(radio_weak, yes)]), rhs(ev(battery_bad, yes), 50)).

rule(rule_4, 'Rule 4',
   lhs([ev(smell_gas, yes), ev(turn_over, yes)]), rhs(ev(problem
    , flooded), 80)).

rule(rule_5, 'Rule 5',
   lhs([ev(turn_over, yes), ev(gas_gauge, empty)]), rhs(ev(
    problem, out_of_gas), 90)).

rule(rule_6, 'Rule 6',
   lhs([ev(turn_over, yes), ev(gas_gauge, low)]), rhs(ev(problem
    , out_of_gas), 30)).
```

Código 6 Representação da base de conhecimento do protótipo

4.2.3.5 Algoritmos a utilizar

Um dos parâmetros mais relevantes para o funcionamento do algoritmo inicial é o grau mínimo de confiança que é indicado no início da utilização do sistema, na página inicial. Se o valor indicado para este parâmetro for demasiado elevado, poderá inibir algumas regras, impedindo o seu disparo no caso de não ser atingido o grau de confiança necessário. Por outro lado, se o valor introduzido for demasiado baixo, permitirá que qualquer regra dispare independentemente do seu grau de confiança, traduzindo-se numa maior árvore de pesquisa.

O motor de inferência é executado mediante o recurso a um predicado principal. Este predicado será o responsável pelo início do processo de inferência. De seguida serão apresentados alguns dos algoritmos considerados mais relevantes. Os algoritmos 1, 2 e 3 representam, através da utilização de pseudo-código, alguma da lógica implementada no motor de inferência.

Algoritmo 1: Algoritmo principal do motor de inferência

```
begin tryToProve
    while empty ControlObj do
        obtemRegra (r);
        disparoPremissas ← true;
        for premissa ∈ LHS do
            ControlObj,disparou ← verificaPremissa (p);
            disparoPremissas ← disparoPremissas && disparou;
            if not empty ControlObj then
                termina o algoritmo;
            end
        end
        if empty ControlObj then
            if disparoTodasPremissas then
                fireRules;
            end
        end
    end
end
```

Algoritmo 2: Verifica se uma premissa é verdadeira

```
input  : Premissa p
output : ControlObj, disparou
begin verificaPremissa
    disparou ← false; ControlObj ← null;
    if checkExistFact (p) ou checkExistQuestionFact (p) ou
    checkExistRuleFact (p,ControlObj) then
        disparou ← true; exit;
    end
end
```

Algoritmo 3: Dispara as regras que possam ser disparadas

```
begin fireRules
    for regra ∈ basedeconhecimento do
        if regra not fired then
            if regra pode disparar then
                calcula grau de confiança;
                dispara regra;
                if regra dispara solução final then
                    termina execução preenchendo ControlObj;
                end
                if regra não dispara solução final then
                    continua;
                end
            end
        end
    end
end
```

4.2.4 Resultado final

Após a implementação do protótipo inicial do motor de inferência e dos testes necessários à verificação do bom funcionamento do mesmo, chegamos ao resultado final que foi utilizado no wexsys.

Este novo motor de inferência foi desenvolvido levando em consideração todas as características consideradas essenciais, tanto para a execução num ambiente multi-utilizador como para a execução num servidor *web* e que foram já descritas nas secções anteriores. De referir ainda que, face ao motor de inferência do sistema base, este apresenta uma série de melhorias tais como rapidez de funcionamento e abstracção da base de conhecimento utilizada.

4.2.4.1 Arranque da execução

Como já referido, o motor de inferência segue os algoritmos descritos na secção 4.2.3.5 e nesta secção será abordado o processo utilizado desde o arranque do motor de inferência até à apresentação dos dados, como podemos ver na figura 19.

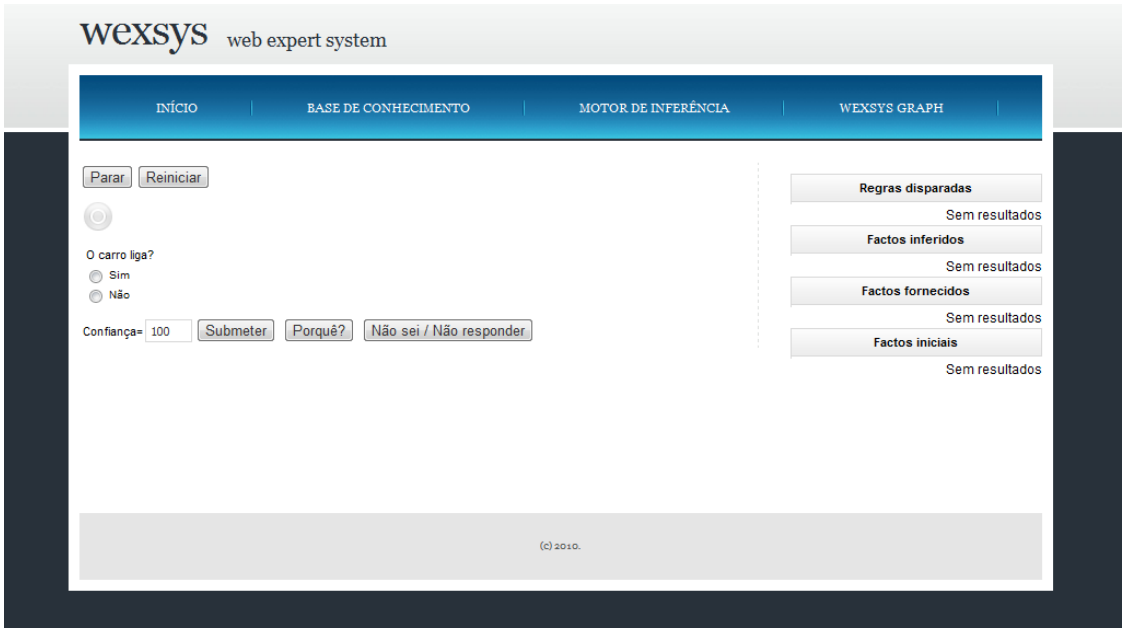

Figura 19 Página inicial do motor de inferência

O utilizador começa na página inicial e selecciona a opção "Motor de Inferência". Depois, é apresentada uma página que permite iniciar o motor de inferência utilizando a

opção “iniciar”. Isto despoleta o mecanismo de inferência como podemos verificar na figura 20.

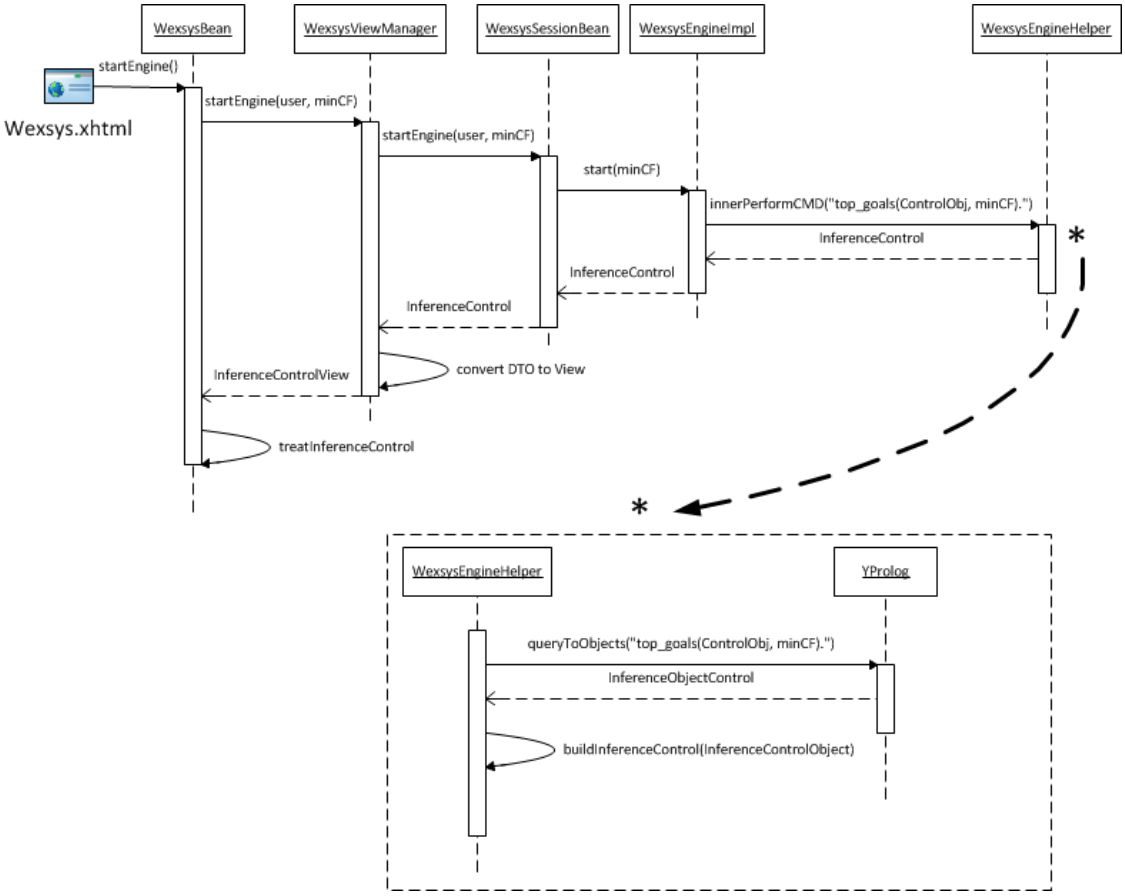

Figura 20 Fluxo de invocação inicial do processo de inferência

A classe YProlog invoca o predicado *top_goals* que mais não é que o ponto de entrada no motor de inferência. A sua função será obter qual a “conclusão final a procurar” (este tópico será descrito na secção 4.3.2) para se poder iniciar o algoritmo propriamente dito. Na listagem de código 7 é possível consultar os predicados responsáveis pela implementação dos algoritmos. Apenas estão listados os predicados principais.

É possível consultar o código do motor de inferência no anexo A.

```
top_goals(ControlObj, MinCF) :-
  top_goal(TopGoal),
  top(TopGoal, ControlObj, MinCF).
```

```
4
top(TopGoal, ControlObj, MinCF) :-
6   retractall(hist/2), set(len, 0),
    getRuleMoreRelevant(TopGoal, Rule),
8   tryToProve(Rule, ControlObj, MinCF).

10 tryToProve(Rule, ControlObj, MinCF):-
    rule(Rule, _, lhs(LHS), _),
12  proveListHyp(LHS, ControlObj, MinCF),
    get(len, LEN), LEN1 is LEN + 1,
14  assert(hist(LEN1,Rule)),
    set(len, LEN1),
16  (
      % check if rule has return control object
18    (var(ControlObj)) ->(
        % no control object found, check if other rules needs to
          be fired
20      fireRules(ControlObj, MinCF)
      );(
22      % control object found
        true
24    )
    ).
26
proveListHyp([], ControlObj, MinCF).
28 proveListHyp([H|T], ControlObj, MinCF):-
    proveHyp(H, ControlObj, MinCF),
30  (
      (var(ControlObj)) -> (
32      proveListHyp(T, ControlObj, MinCF)
      );(
34      true
      )
36  ).


38
%fact already known
40 proveHyp(H, ControlObj, MinCF):-
```

```
    H = ev(X, Y), fact(X, Y,_ ), !.

% there is a question that can lead to a fact
proveHyp(H, ControlObj, MinCF):- H = ev(X, _),
 not(fact(X, _, _)),
 question(X, _, _, _),
 not(not_question(X)),
 ControlObj = ask(X), !.

%there is a conclusion that can
%lead to a question that can lead to a fact
proveHyp(H, ControlObj, MinCF):- H = ev(X, _),
 rule(Rule, _, lhs(LHS), rhs(ev(X,_), V)),
 tryToProve(Rule, ControlObj, MinCF), !.
```

Código 7 Predicados principais do motor de inferência

4.2.4.2 Mecanismo de respostas

O mecanismo de respostas é algo similar ao processo descrito anteriormente. A sua função será guardar a resposta que foi seleccionada pelo utilizador e prosseguir com o algoritmo de forma a que uma nova questão seja colocada ou a solução final alcançada.

Um parâmetro considerado obrigatório na submissão de uma resposta é o grau de confiança. Este valor servirá para realizar o cálculo do grau de confiança da regra quando a mesma puder ser disparada.

Podemos ver na figura 21 o processo de resposta a uma questão.

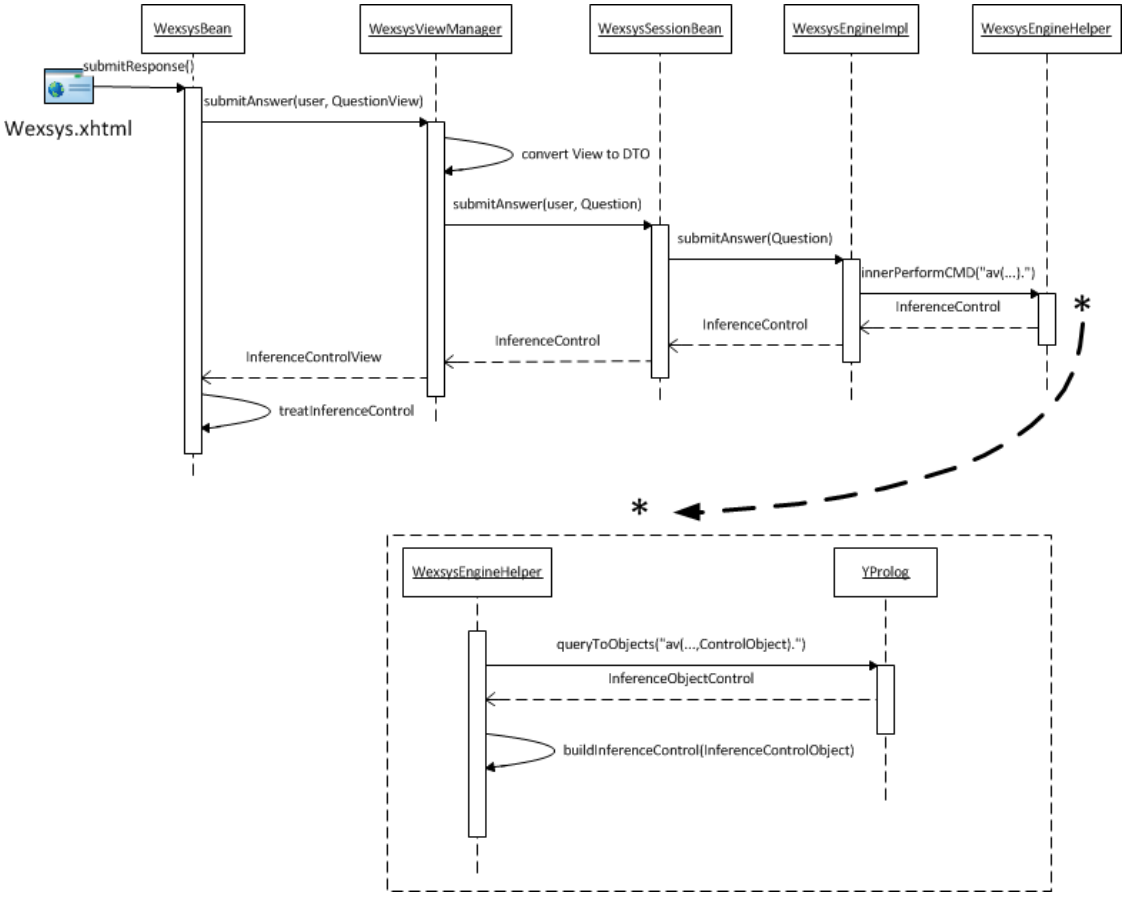

Figura 21 Fluxo de invocação de resposta a uma questão

É ainda possível não responder à questão colocada utilizando a opção "não sei / não responder". Esta opção irá guardar internamente uma indicação para não voltar a colocar a questão ignorada evitando entrar em ciclos.

À medida que o processo avança é possível acompanhar o estado actual do motor de inferência. Este acompanhamento é realizado mediante a análise da zona situada do lado direito da página e que vai sendo actualizada a cada interacção com o motor de inferência. A zona referida permite obter informações acerca de:

- Regras que já foram disparadas;
- Factos que foram inferidos pelo disparo das regras;
- Factos que foram facultados pelo utilizador ao responder às questões;
- Factos conhecidos antes do início do processo de inferência.

Estes factos são obtidos através do predicado "inf_sys((L_Rules, L_Inf_Facts, L_Know_Facts, L_Init_Facts)". Este predicado é invocado após cada interacção com o motor de inferência e desta forma os dados estão sempre actualizados.

4.2.4.3 Conclusão final encontrada

Quando o disparo de uma regra leva a que seja adicionado um facto que corresponde a uma solução final, o motor de inferência termina a sua execução devolvendo a conclusão final a que chegou bem como o respectivo grau de confiança. Na figura 22 podemos ver a fase final do processo de inferência. O grau de confiança da conclusão final é atribuído com recurso à fórmula seguinte:

Dada uma regra X com premissas P_1 a P_n e $CF = CF(X)$, então temos:

$CF(X;P_1,\ldots,P_n) = min[CF(P_1),...,CF(P_n)] \times CF(X)$

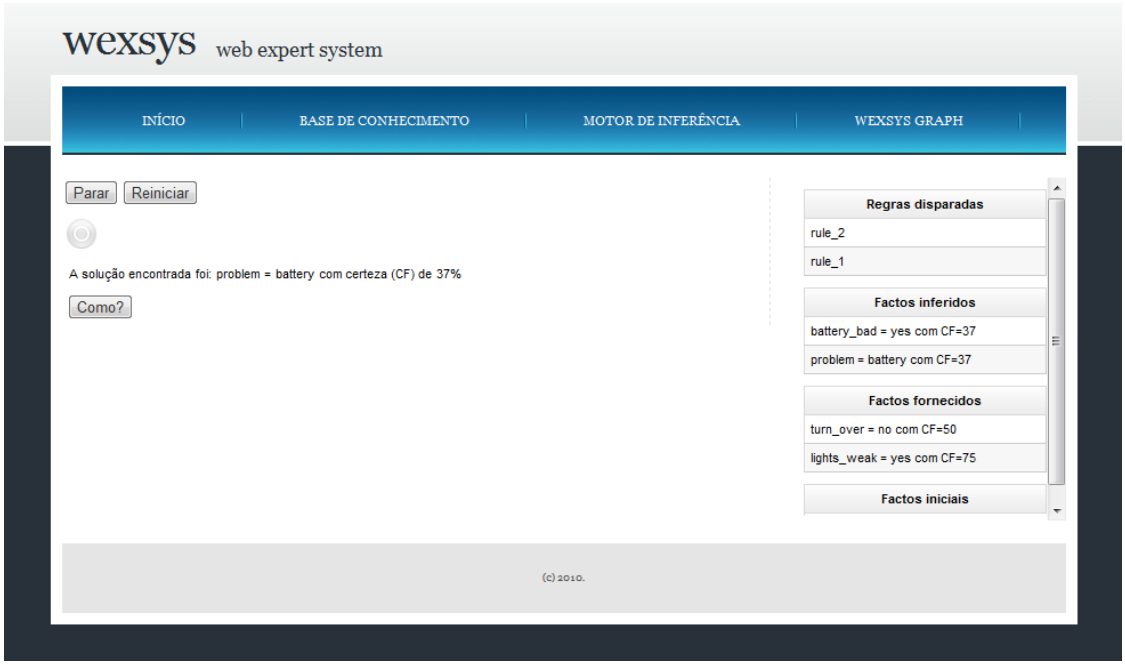

Figura 22 Conclusão da inferência

4.2.4.4 Explicações

É possível, em qualquer altura, colocar a questão "porquê?" antes de submetermos uma resposta ou a questão "como?" após uma conclusão final. Este tipo de explicações contribui para a fase de validação do sistema pericial colocando-as ao dispor do perito

responsável pela validação, expondo desta forma o raciocínio seguido no decorrer da inferência. Exemplos da solicitação de explicações podem ser consultados através das figuras 23, 24 e 25

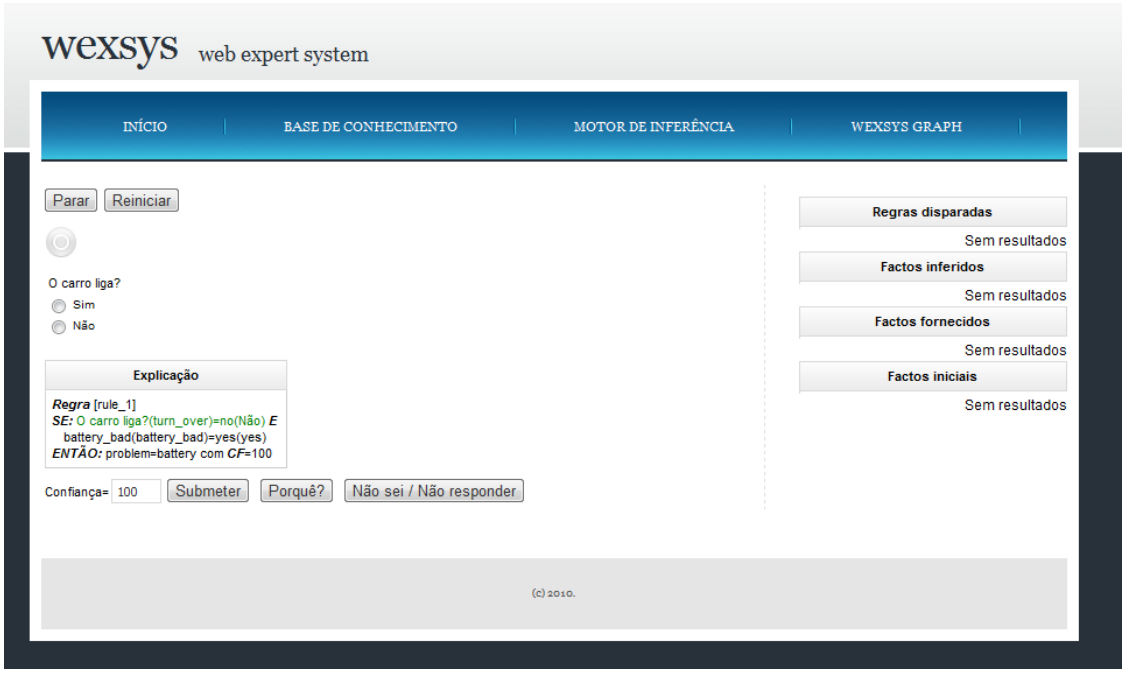

Figura 23 Utilização da explicação "porquê"

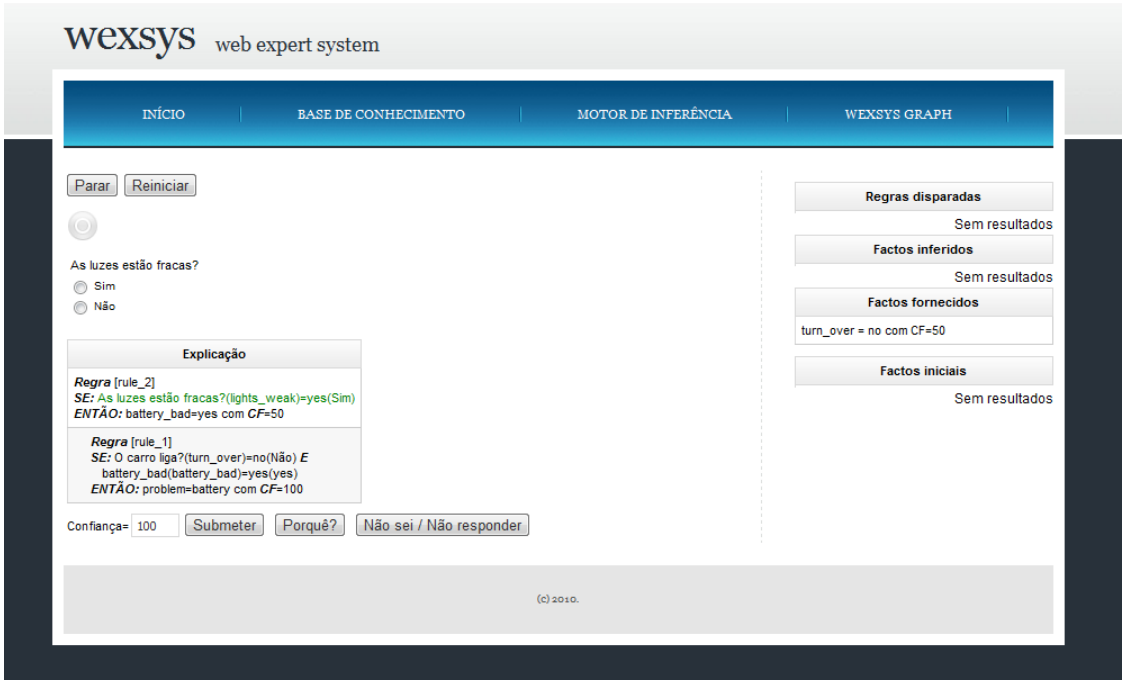

Figura 24 Utilização da explicação "porquê" numa regra intermédia

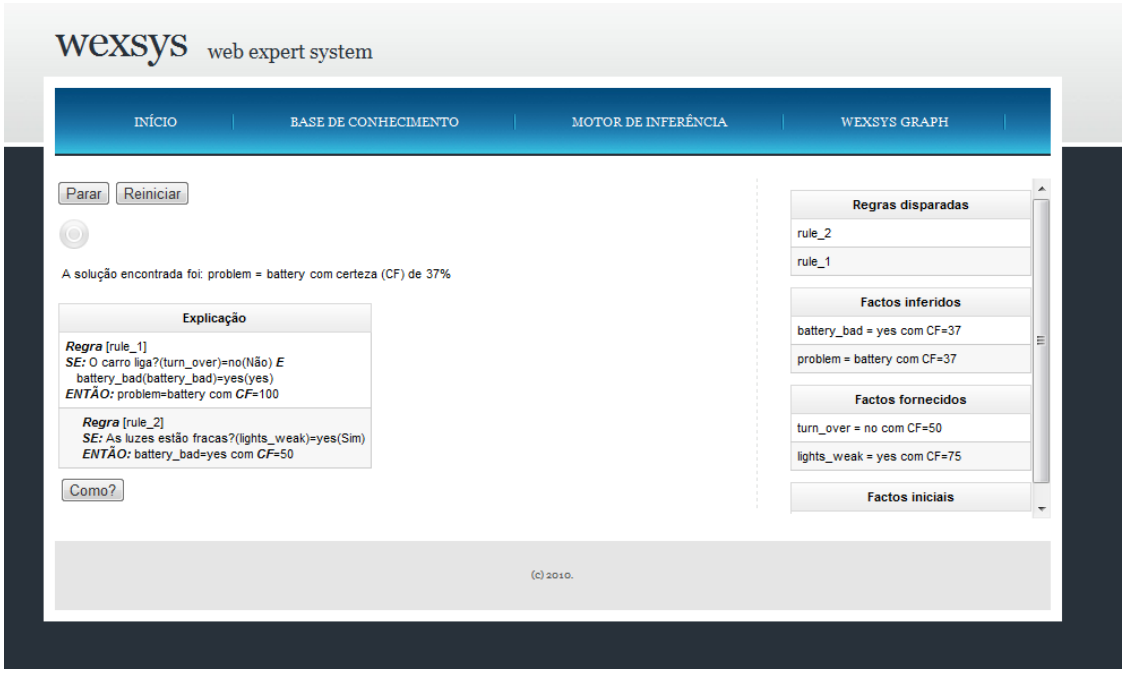

Figura 25 Utilização da explicação "como" numa solução final

As explicações "porquê?" foram implementadas recorrendo a predicados temporários (*hist*) que acompanham o desenrolar do motor de inferência. É possível, em qualquer altura, consultar esses predicados para obter de uma forma imediata a explicação do porquê da questão estar a ser feita.

As explicações "como?" já seguem um mecanismo mais complexo. Quando uma regra é disparada origina conclusões, intermédias ou finais, essas conclusões não são mais do que simples factos. O motor de inferência, sempre que dispara uma regra, associa ao facto a regra que o originou por intermédio do predicado *inf_exp_how*.

Desta forma, basta algum código para que seja possível a obtenção de ambas as explicações. Esse código pode ser consultado na listagem 8 em que os predicados *why* e *how* se referem às explicações "porquê" e "como" respectivamente.

```
why(L):-
 findall(r(O,X), hist(O,X), L).

how(Attr, Value, Result):-
  inf_exp_how(Attr, Value, Rules),
  explainHow(Rules, Explanation),!,
  append(Rules, Explanation, Result).

explainHow([], []).
explainHow([H|T], Result):-
    innerExplain(H, ExplanationH),!,
    explainHow(T, ExplanationRules),
    append(ExplanationH, ExplanationRules, Result).


innerExplain(Rule, Explanation):-
  rule(Rule, _, lhs(LHS), _),!,
  explainLHS(LHS, Explanation).

explainLHS([], []).
explainLHS([ev(Attr, Value)|T], ExplanationA):-
  not(inf_exp_how(Attr, Value, _)),!,
  fact(Attr, Value, _), !,
  explainLHS(T, ExplanationA), !.
```

```
26 explainLHS([ev(Attr, Value)|T], ResultA):-
     how(Attr, Value, ExplanationA),!,
28   explainLHS(T, Explanation), !,
     append(ExplanationA, Explanation, ResultA).
```

Código 8 Predicados principais do sistema de explicações

4.3 Base de conhecimento

A base de conhecimento armazena o conhecimento extraído de um ou mais peritos e consiste numa série de factos e regras acerca de um determinado domínio. Sendo dependente da área que se pretende abranger com o sistema pericial, a base de conhecimento é de construção específica para cada situação em que seja necessário desenvolver este tipo de sistema.

Os procedimentos de inserção, actualização e remoção dos factos e regras que compõem a base de conhecimento (gestão da base de conhecimento) terão que ser, na medida do possível, procedimentos simples que sejam facilmente compreendidos pelos utilizadores.

As funcionalidades de gestão da base de conhecimento estão disponíveis na opção "Base de Conhecimento" da página inicial como podemos observar na figura 26.

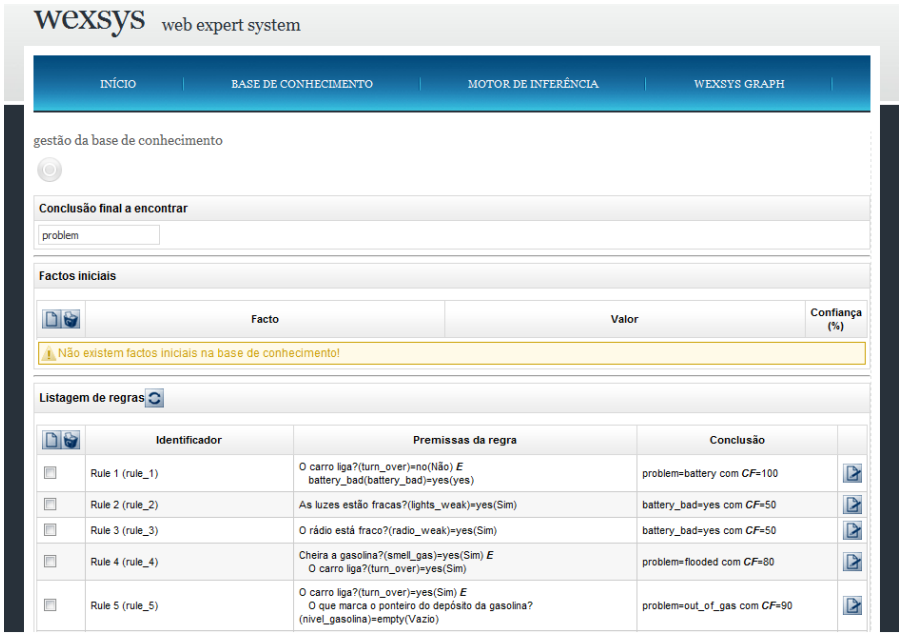

Figura 26 Página inicial da gestão da base de conhecimento

4.3.1 Leitura da base de conhecimento

Ao iniciar a execução, a base de conhecimento é carregada para memória a partir de um ficheiro. Este processo é realizado de forma iterativa e de acordo com uma determinada sequência: regras, questões, conclusão final e por último os factos iniciais.

Por sua vez, o carregamento das regras é efectuado em duas fases sendo que, primeiro são obtidas as regras existentes ("retrieve_rules(L)") e depois para cada regra é construída a sua estrutura ("buildRuleObject"). Podemos observar a descrição deste processo na figura 27.

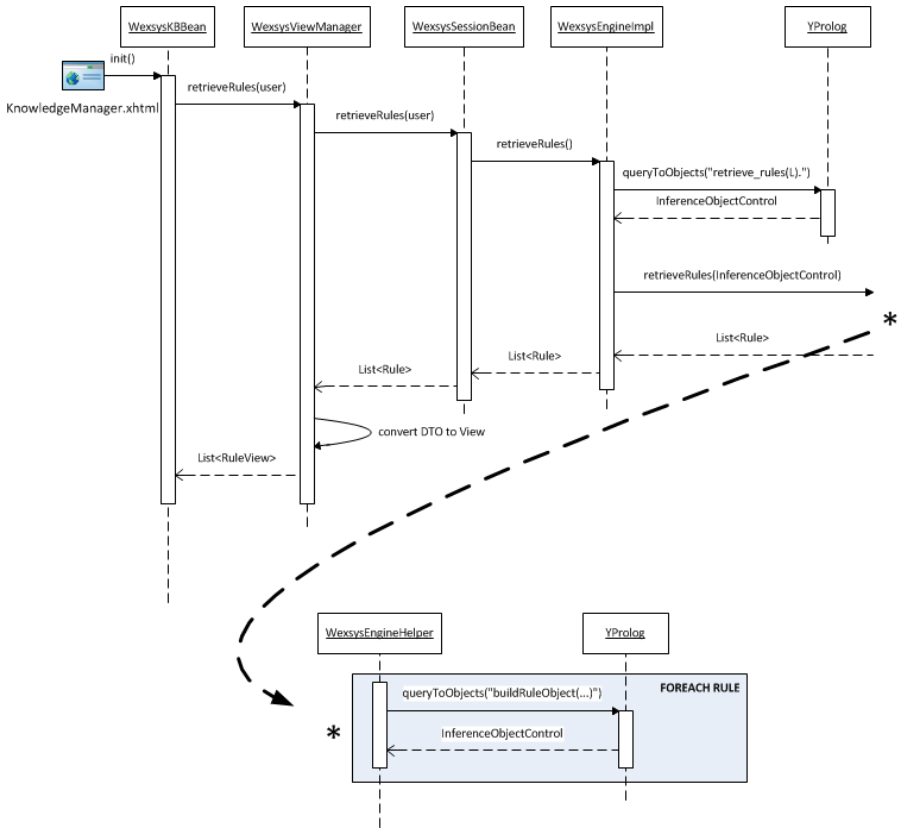

Figura 27 Processo de carregamento das regras

Quanto ao carregamento das questões, é similar ao das regras já que é efectuado igualmente em duas fases. Primeiro obtêm-se todas as questões (“retrieve_questions(L).”) e depois, para cada questão, é preenchida a sua estrutura (“buildQuestionObject”).

Já o carregamento da conclusão final é realizado através do predicado “top_goal(L)”. Este predicado vai devolver a conclusão final guardada.

Por fim, os factos iniciais são obtidos reutilizando um processo de obtenção de factos (“inf_sys((L_Rules, L_Inf_Facts, L_Know_Facts, L_Init_Facts)”). Este processo já foi referido na secção que descreve o motor de inferência (4.2.4.2).

4.3.2 Conclusão final a encontrar

A conclusão final a encontrar representa o predicado que vai ser considerado como objectivo no decorrer do motor de inferência. Só vão ser consideradas soluções que convergem para uma solução final que contenha o predicado especificado no RHS.

Para definir uma conclusão final basta especificar o nome do predicado da solução que pretendemos encontrar com o motor de inferência.

4.3.3 Factos iniciais

Os factos iniciais permitem adicionar conhecimento prévio a uma base de conhecimento. Para adicioná-los é necessário utilizar o botão "Adicionar facto inicial" e preencher os campos necessários. Para remover um facto inicial previamente inserido, é utilizado o botão "Remover facto inicial" e é necessário confirmar a acção que se está a efectuar de forma a evitar uma remoção acidental.

4.3.4 Regras

As regras são o ponto essencial da gestão da base de conhecimento. As regras são os dados disponíveis para a aplicação, e como tal, quanto melhor forem definidas, melhor qualidade a aplicação terá no sentido em que os resultados serão mais correctos e fiáveis.

4.3.4.1 Criar/Alterar uma regra

O processo de criação de uma nova regra foi simplificado. Já não é necessário conhecer a sintaxe das regras e também já não é necessário conhecer os factos a utilizar como premissas.

Para adicionar uma regra apenas temos que arrastar questões e/ou conclusões intermédias de forma a construir a regra que pretendemos inserir.

No caso de estarmos a alterar uma regra, ou no caso de termos adicionado uma questão ou conclusão intermédia incorrectamente, podemos eliminar essa premissa simplesmente seleccionando a "*checkbox*" respectiva e utilizar a opção "Remover seleccionados".

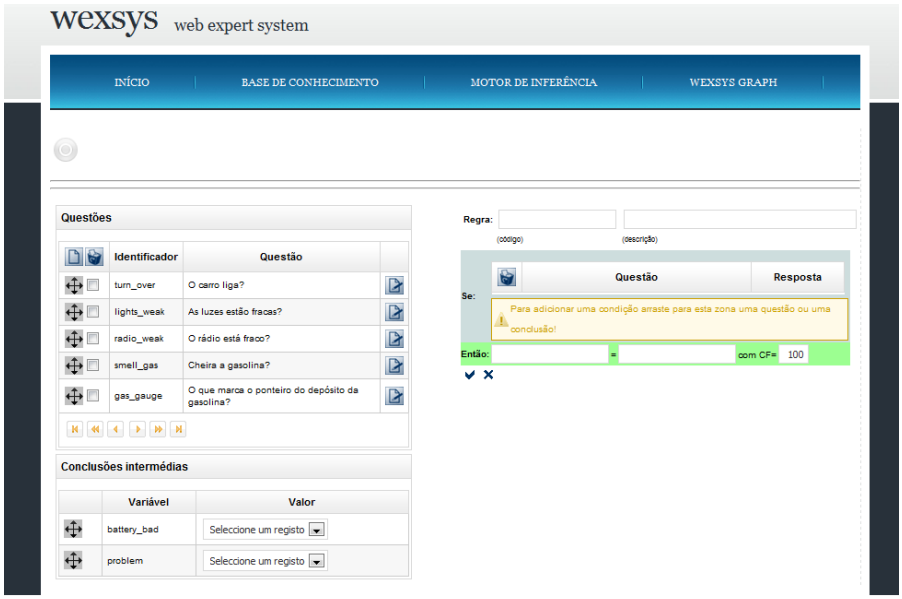

Figura 28 Introduzir uma nova regra

Após termos arrastado todas as questões e/ou conclusões intermédias para a lista de premissas, temos que indicar os respectivos valores. Este processo está simplificado no sentido em que apenas temos que utilizar a caixa de selecção à frente de cada controlo e seleccionar o respectivo valor.

Os valores apresentados são obtidos de forma diferente quer se trate de uma questão ou de uma conclusão intermédia. No caso das questões, os valores obtidos são os indicados aquando da sua inserção. No caso das conclusões intermédias, são obtidos todos os valores utilizados nas regras até ao momento. Sempre que uma regra é inserida ou actualizada é despoletado um processo que actualiza os valores das conclusões intermédias com os novos valores introduzidos.

Figura 29 Apresentação de valores configurados na questão

4.3.4.2 Questões

As questões acabam por ser o ponto de contacto da aplicação com o utilizador. Sempre que é necessário suspender o processamento para questionar o utilizador, este terá a hipótese de intervir na execução. Através das questões colocadas, vai ser possível inferir outros factos que, no final e numa situação ideal, conduzem a uma solução possível.

Uma questão é identificada por um código, por uma descrição e ainda por um conjunto de valores possíveis, as respostas. Quando inserimos uma nova regra podemos criar novas questões utilizando a opção "Adicionar uma questão". Surge então o formulário de introdução de uma questão como podemos ver na figura 30.

Figura 30 Formulário de introdução de uma questão

As respostas configuradas no formulário de introdução serão as respostas que irão aparecer quando inserirmos uma nova regra e arrastarmos a questão para a lista de premissas.

4.3.4.3 Verificação

Após a introdução ou alteração das regras, deve ser realizado o processo de verificação da base de conhecimento. De uma forma quase imperceptível, na fase de introdução de regras, estamos a activar este processo já que é garantido que as regras introduzidas estão coerentes e que os valores das premissas efectivamente existem. Podemos dizer que estamos perante a primeira fase do processo de verificação designado por verificação pró-activa.

Para este trabalho foi considerado a verificação de dois tipos de anomalia na base de regras: redundância e regras duplicadas. A inclusão de novos mecanismos de verificação é possível bastando para isso a alteração do predicado "verification(L)".

A grande diferença entre os dois métodos implementados é que uma regra redundante adiciona conhecimento desnecessário à base de conhecimento enquanto que uma regra duplicada, como o nome indica, é apenas uma duplicação do conhecimento.

Na figura 31 podemos verificar o mecanismo de verificação implementado que permite identificar que regra está em conflito com que outra regra e qual o tipo de conflito.

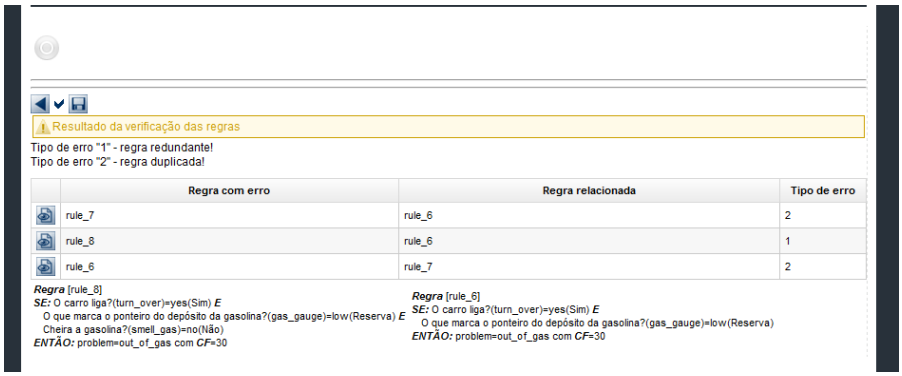

Figura 31 Verificação da base de conhecimento

4.3.5 Gravação da base de conhecimento

O processo de gravação da base de conhecimento é similar ao processo de leitura com a diferença de não ser iterativo. O processo de gravação da base de conhecimento inicia com a acção do utilizador na opção "Finalizar" que invoca o método "saveAll()" do WexsysKBBean. O resto do processo pode ser observado na figura 32.

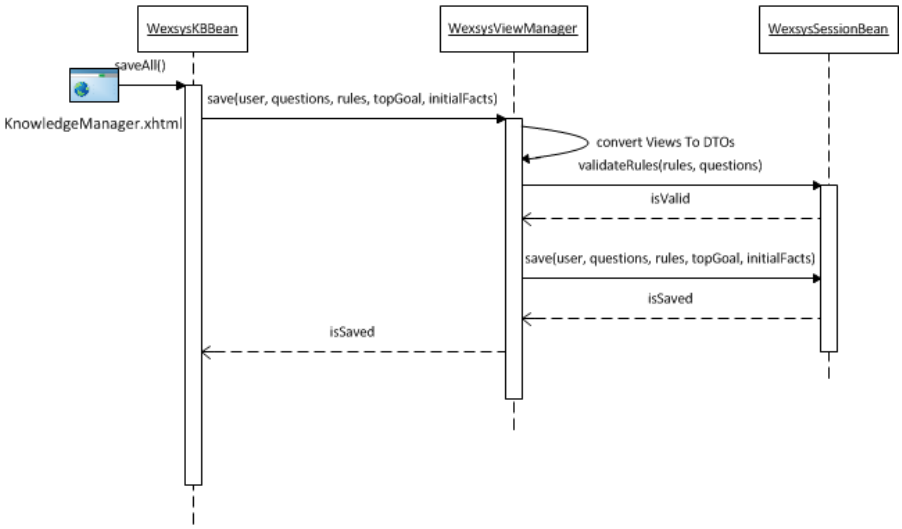

Figura 32 Processo de gravação da base de conhecimento

4.3.6 Conversão GISPSA

A conversão da base de conhecimento do GISPSA para o formato wexsys foi efectuada em duas fases. Na primeira fase, foi feita uma migração sintáctica da forma de representação de regras utilizada para a forma utilizada pelo wexsys. Na segunda fase, foi efectuado um processo manual de conversão.

O sistema anterior delegava nas regras alguns predicados utilizados unicamente para o sistema de explicações. Esta característica tornou a migração algo morosa já que era necessário proceder à análise regra a regra de quais os predicados necessários e quais os descartáveis.

4.4 Análise gráfica

Para além do motor de inferência e da gestão da base de conhecimento, foi implementado, como prova de conceito, um sistema gráfico que permite observar facilmente a representação de uma base de conhecimento sob a forma de um grafo.

Este sistema permite visualizar e consultar a explosão combinatória das regras e das premissas que as compõem. Este tipo de ferramenta pode ser bastante útil quando utilizado nas tarefas de validação por parte do perito. Desta forma, o perito consegue, de uma forma visual e simples, verificar o relacionamento entre as conclusões e as premissas.

4.4.1 Implementação

Esta funcionalidade foi implementada recorrendo à utilização de um controlo externo, designado por *TGGraphLayout* e desenvolvido por Alexander Shapiro (2010). O controlo disponibiliza uma *applet* que encapsula funcionalidades que permitem a interacção gráfica com a representação de um grafo.

Por se tratar de uma *applet*, a comunicação com os *managed beans* torna-se, tecnicamente, mais difícil de implementar. Esse motivo levou à necessidade de criação de uma *servlet* para simplificar o processo de interacção entre a aplicação e a *applet* do controlo visual.

A *servlet WexsysServlet* foi criada para dar suporte à comunicação com o *managed bean WexsysKBBean*, sendo este responsável pela obtenção dos dados no formato reconhecido pelo controlo.

O controlo teve que ser alterado de forma a permitir a interacção com a *servlet* bem como efectuar o tratamento dos dados no formato especificado, ou seja, criar uma espécie de protocolo de comunicação.

O algoritmo 4 resume a implementação efectuada ao nível da *applet* e o algoritmo 5 demonstra de que forma esses dados são obtidos.

4.4.2 Funcionamento

O grafo resultante é construído a partir de uma posição inicial designada por ponto de partida. É utilizada a designação "@@@@@" para simplificar a identificação no grafo e é possível ver o resultado da página inicial na figura 33.

As conclusões são representadas por rectângulos com um fundo branco e as premissas são identificadas por rectângulos com fundo azul.

Algoritmo 4: Algoritmo responsável pelo desenho da applet

```
input  : graphPaths, graphConclusions
begin drawGraph
    initializeNodes ();
    for path ∈ graphPaths do
        init old;
        for step ∈ path do
            if nodes do not contain step then
                buildNode (step);
            end
            if step is first then
                old ← step;
            else
                addEdge (old, step);
                old ← step;
            end
        end
    end
    createStartNode ();
    linkUnlinkedNodesToStartNode ();
end
```

O ponto de partida liga-se então às regras que apenas contenham premissas que sejam questões, construindo o grafo de possibilidades. Após identificar as regras que são representadas com ligação ao ponto de partida, são tratadas as restantes regras, construindo o resto do grafo à medida que as regras são avaliadas.

É possível centrar o grafo numa premissa ou numa conclusão. Ao centrarmos o grafo são mantidos apenas os nós que se ligam directamente ao nó que seleccionámos. No entanto é possível expandir um nó sem este tipo de comportamento bastando para isso utilizar as funcionalidades disponíveis ao pressionar a tecla direita do rato sobre o

Figura 33 Página inicial da análise gráfica

Algoritmo 5: Algoritmo para construção dos dados do grafo

```
output : graphPaths and grapConclusions
begin retrieveGraphPathAndConclusions
    questions ← gatherQuestion;
    for rule ∈ rules do
        isFinalRule ← conclusaoFinal igual RHS de rule;
        if isFinalRule then
            add rule to grapConclusions;
            simpleRule ← true;
            graphPaths ← null;
            for premissa ∈ ruleLHS do
                isConditionAskable ← questions contains premissa;
                if isConditionAskable then
                    add premissa and value to graphPaths;
                else
                    simpleRule ← false;
                    retrieveGraphFromAtom (premissa);
                    add result to graphPaths;
                end
            end
        end
    end
end
```

respectivo nó. As funcionalidades disponíveis são: expandir, colapsar, esconder e centrar e são auto-descritivas pelo que dispensam qualquer explicação do seu funcionamento.

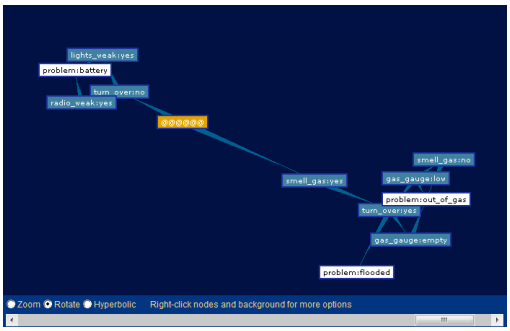

Figura 34 Exemplo de grafo totalmente expandido

É possível utilizar esta funcionalidade para identificarmos um cenário específico, por exemplo para consultar quais as premissas necessárias para a obtenção de uma determinada solução.

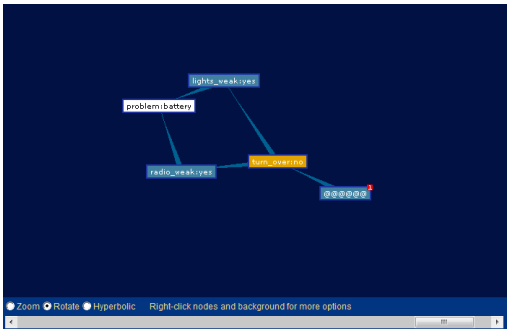

Figura 35 Exemplo de utilização para identificação de um cenário concreto

4.4.3 Limitações

Esta funcionalidade, no seu estado actual, apresenta algumas limitações. Convém referir antes de mais que esta solução não é uma alternativa ao motor de inferência mas sim uma ferramenta de apoio visual auxiliar ao mesmo.

Das limitações existentes salientam-se as seguintes:

- a utilização deste controlo obriga a que o utilizador esteja familiarizado com as premissas e os seus valores. Não estão disponíveis os textos de apoio para as premissas da mesma forma que são mostrados no motor de inferência. Uma possível solução passaria pela interacção da *applet* com a *servlet* para, através do mecanismo de *ajax push*, despoletar a actualização da JSP mostrando o texto do elemento seleccionado;

- o tamanho dos nós é grande quando comparado com o espaço disponível no controlo, o que por vezes obriga a algumas sobreposições dos nós, tornado-o imperceptível. Este problema pode ser observado na figura 34. Para atenuar este problema existem controlos que permitem modificar o factor de ampliação, rodar o controlo e controlar o grau de visualização hiperbólica;

- após a navegação no grafo, perde-se de alguma forma o contexto que o originou. Este problema pode ser observado na figura 35.

5
Conclusões

Até que ponto é que um software consegue efectivamente substituir um ser humano quando estamos a falar de raciocínio? Se esse raciocínio for estritamente linear, seguindo um determinado percurso com alguns desvios ao longo do mesmo que acabam por ditar um desfecho diferente consoante as decisões que vão sendo tomadas, a substituição pode mesmo ser efectiva e até mesmo melhor em alguns casos.

Um perito não é no entanto alguém que se limita a dizer sim ou não ou alguém que apenas decide se é necessário seguir por este caminho ou aquele. Um perito normalmente observa dados e factos e infere sobre os mesmos tendo a sensibilidade para, em muitas ocasiões, conseguir até mesmo obter a solução final muito antes de terminar a análise de todos os parâmetros. A isso chama-se conhecimento heurístico não linear, afectada por uma enormidade de factores que apenas é possível encontrar num ser humano.

Estes factores podem ter tanto de benéfico como de nocivo. Ao não analisar a totalidade dos dados, um perito poderá incorrer num erro ou, mesmo tendo-os analisado até ao fim, poderá não ter estado completamente atento ou compenetrado no que estava a fazer pelo mero facto de poder estar cansado ou indisposto para executar tal tarefa naquele preciso momento.

Um sistema pericial pode realmente ser uma mais valia no sentido em que está totalmente desprovido de emoções e fadiga, ao contrário de um ser humano. Para além disto, não é necessário temer que o sistema pericial se desloque a uma qualquer localização remota e de difícil acesso ou que chegue fora do horário previsto, limitado pelos atrasos de uma simples viagem.

Todos estes factos podem realmente ser considerados quando analisamos o funcionamento do GISPSA. Efectivamente trata-se de um sistema pericial, está sempre disponível e totalmente desprovido de características humanas que poderiam eventualmente toldar o raciocínio do mesmo. No entanto, este sistema carecia de algo muito importante, a

acessibilidade a partir de qualquer local, em qualquer hora e por parte de vários utilizadores em simultâneo de modo imediato e sem pré-aviso. Antes de mais, o GISPSA é uma aplicação de execução local e não foi de todo desenvolvida para funcionamento em rede. Como tal, o acesso à mesma estava à partida bastante limitado. Depois, cada instalação apenas permitia a interacção com um utilizador de cada vez. Existem ainda algumas limitações identificadas no GISPSA que foram ultrapassadas com o desenvolvimento do wexsys, tais como: manutenção da base de conhecimento, capacidade de raciocinar sobre conhecimento incerto e a implementação de métodos de verificação.

Hoje em dia é possível aceder a uma rede global (a internet) a partir de qualquer ponto do planeta e isto torna igualmente possível disponibilizar uma aplicação de forma a que qualquer pessoa a consiga utilizar a qualquer hora e a partir de qualquer local. O wexsys aplica essa transformação ao GISPSA e leva-o ao próximo patamar evolutivo.

Como qualquer evolução, também esta pretende não só dotar a nova versão com funcionalidades que a anterior não possuía como também pretende melhorar e corrigir os problemas que eventualmente foram encontrados. Ao longo do tempo de utilização do GISPSA, verificaram-se algumas coisas que podiam e deviam ser substituídas. Além disto, a própria evolução dos sistemas gráficos e de interacção colocaram o GISPSA como uma aplicação de visual já algo ultrapassado, aspecto que também carecia de uma revisão para a realidade mais actual.

A opção pela incorporação do wexsys num portal teve a ver com as grandes capacidades que este tipo de infra-estuturas tem hoje em dia. Impulsionados por empresas como a IBM, os portais *web* representam janelas de negócio com ligação ao mundo e com as potencialidades que daí advêm. Características como a facilidade de acesso, sistemas de *backup*, manutenção de tempos de *uptime* elevados, rapidez de funcionamento, facilidade de funcionamento e leitura, entre muitas outras, representam uma mais valia para este tipo de alojamento de aplicações.

5.1 Contributos da solução desenvolvida

O grande objectivo a que este trabalho se propunha foi alcançado e consistia na criação de um sistema pericial baseado no GISPSA mas que pudesse ser acedido a partir de uma rede, tornando-o global. Os restantes objectivos foram igualmente alcançados e o resultado final mostra-se sólido e fiável para candidato efectivo à substituição do sistema que lhe dá a base. No entanto, existem outros pontos positivos na solução encontrada quando comparada com outros sistemas periciais já referidos nos capítulos anteriores.

Esses sistemas são o LOMA, o MDSS e o Whale watcher.

Ao proceder a uma comparação com estes sistemas, a solução desenvolvida neste trabalho apresenta algumas vantagens. Desde logo, estamos a falar de aplicações *web* com períodos de desenvolvimento diferentes, o que implica a utilização de tecnologias diferentes. A vantagem da utilização de técnicas de programação mais recentes, dinâmicas, eficazes e visualmente mais apelativas, coloca o wexsys num patamar efectivamente superior. Para além disso, o wexsys pode (como resultado da utilização dessas mesmas tecnologias) ser facilmente integrado noutras aplicações como *webservices* ou portais enquanto que os outros sistemas não.

Outra das particularidades dos sistemas estudados é que não possuem (de acordo com as respectivas documentações) mecanismos de verificação de regras e sempre que é necessário proceder a alterações a essas mesmas regras, o utilizador terá que ter um conhecimento profundo acerca da sintaxe utilizada (gramática rígida). Para além disto, também não possuem (com excepção do MDSS) suporte a várias línguas.

Já quanto à questão da capacidade de funcionamento multi-utilizador, apesar de estarmos a falar de aplicações *web*, os sistemas estudados neste documento não estão preparados para cargas de processamento demasiado elevadas porque não foi contemplada a possibilidade de um funcionamento distribuído, o que permitiria aliviar a carga de processos e aumentar o número máximo de utilizadores.

Para descrever mais em detalhe os objectivos alcançados com a solução proposta, passamos a enunciá-los:

- Criação de uma aplicação *web* - Ao criar a nova aplicação, a questão da distribuição dessa mesma aplicação através de uma rede era fulcral já que permitiria uma série de factores que até aqui não estavam disponíveis. A possibilidade de utilização de uma base de conhecimento global, onde todos os utilizadores pudessem contribuir para o seu melhoramento, facilidade de acesso a partir de qualquer localização, disponibilidade a qualquer hora e dia, capacidade de realização de backups para garantir a continuidade dos dados e múltiplos acessos simultâneos, foram questões que ficaram resolvidas com a concretização deste objectivo.

- Verificação das regras - Uma das lacunas do sistema base, o GISPSA, era a falta de capacidade para realizar a verificação das regras que era colocadas na base de conhecimento. Era possível chegar a cenários de inconsistência através da contradição de regras, podendo chegar a cenários em que simplesmente não era possível prosseguir com a inferência das regras nem tão pouco chegar a uma conclusão ou solução possível.

Ao adicionar um mecanismo de verificação de regras como o foi neste caso, assistimos não só ao cumprimento de mais um objectivo mas também a uma evolução significativa do novo sistema pericial que, com esta característica, se torna mais fiável e eficaz.

- Manutenção da base de conhecimento - Neste novo sistema, é mais simples proceder à actualização da base de conhecimento, quer para adicionar, alterar, ou remover factos na mesma. Já não é necessário ter formação sobre a sintaxe necessária para realizar estas operações, deixando de lado a rigidez que era necessária aplicar no sistema base. Isto permite alargar o leque de utilização do sistema a outros utilizadores que, na versão anterior, não teriam oportunidade para o fazer.

- Melhor usabilidade - Uma das grandes transformações para quem conhecia o sistema anterior e encontra a nova versão, passa pelo aspecto visual. Quando um utilizador é confrontado com uma aplicação, aquilo com que se depara é a parte de interacção responsável por mediar o diálogo entre o utilizador e a aplicação per si.

  Quando uma aplicação, qualquer que ela seja, possui um visual pouco atraente e apelativo para o utilizador, com botões e janelas pouco interessantes, componentes mal colocados, atalhos escondidos e de difícil percepção, ou até mesmo com cores pouco agradáveis, o utilizador acaba por, seja de forma gradual ou imediata, abandonar a aplicação e mostra pouco interesse em voltar a utilizá-la a não ser que seja efectivamente obrigado a fazê-lo.

  Neste trabalho, através da utilização de técnicas de programação *web* tais como *Javascript* e *Ajax* e ainda através da utilização do *Icefaces*, foi possível chegar a um visual considerado por muitos utilizadores como interessante e actual o que vai plenamente de encontro ao objectivo proposto para este ponto.

- Multi-utilizador - Uma das grandes críticas do sistema base tinha efectivamente a ver com o facto de não ser possível colocar mais do que um utilizador a trabalhar ao mesmo tempo. Para tal ser possível, seria necessário proceder à instalação do sistema noutro computador e consequentemente, adquirir outra licença comercial de utilização.

  Ao adicionar a capacidade de vários utilizadores poderem estar a trabalhar no sistema em simultâneo, este objectivo verifica-se como estando cumprido na totalidade pois não apenas isso já é possível neste novo sistema pericial, como a capacidade de processamento não é afectada já que é possível proceder à colocação

da aplicação num sistema de *clustering* em caso de défices ou carga excessiva de processamento.

- Sistema de explicações - A solução proposta contém um sistema de explicações que pode ser utilizado para obter explicações a perguntas do tipo "como" e "porquê". Este ponto não representa uma melhoria mas sim a manutenção de uma funcionalidade que o GISPSA já disponibilizava.

- Análise gráfica - Este módulo é uma nova abordagem à análise de uma base de conhecimento que não é normalmente utilizado nos sistemas periciais conhecidos e que possibilita uma análise visual facilitadora do comportamento e funcionamento da nova solução.

- Suporte a várias línguas - A solução proposta permite de uma forma bastante simples a implementação de novas linguagens. A parte da solução que envolve o sistema pericial e a parte da aplicação *web* estão preparados para esta tipo de suporte contudo, a base de conhecimento não contempla este mecanismo e teria que ser implementado.

- Arquitectura - O tipo de arquitectura da solução proposta permite uma simples integração com outros tipos de aplicação *web* como por exemplo *webservices* ou portais. Possibilita ainda, como já referido, a separação da aplicação de uma forma modular permitindo uma distribuição ao nível do processamento.

- Migração - No processo de migração da base de conhecimento foram detectados e corrigidos alguns problemas. Regras inconsistentes, regras repetidas e regras que não trazem novo conhecimento foram adaptadas e resolvidas, evoluindo a nova base de conhecimento para uma entidade mais fiável e consistente.

5.2 Limitações e trabalho futuro

Este trabalho representa um esforço no sentido de apresentar uma versão melhorada do sistema GISPSA e apesar de o conseguir em muitos aspectos e essencialmente em todos aqueles que foram inicialmente propostos, ficam alguns aspectos que ainda poderão ser implementados futuramente e que certamente contribuirão para um sistema ainda melhor.

Assim, apresentam-se as limitações identificadas e as perspectivas de trabalho futuro:

- Validação de regras - Apesar de ser já possível neste novo sistema pericial proceder à verificação de regras como mecanismo de prevenção de redundâncias e contradições, ainda não é possível a validação por parte de um perito dessas mesmas regras a não ser que essa validação seja feita em modo *ad hoc*.

  Seria importante a criação de um modo "‘supervisor"’ ou modo "‘perito"’ para que a aplicação pudesse adoptar uma postura não só de linguagem adaptada a um utilizador com maior formação na área (como será o caso de perito) mas também pudesse assumir uma postura de simulação em que seriam apresentados não só os passos tomados como eventuais cenários em caso de respostas diferentes. Isto permitiria a um perito confrontar o seu próprio conhecimento com o sistema pericial, realizando desta forma uma validação do funcionamento do sistema e da base de conhecimento.

  Para além disto, sendo uma aplicação *web*, o modo "‘supervisor"’ ou modo "‘perito"’ poderia ser acedido a partir de qualquer sítio e a qualquer hora. Se o responsável pela validação da base de conhecimento (normalmente o perito que contribuiu com o conhecimento), não estiver disponível para o fazer por motivos de ausência ou incompatibilidade de horários, poderá sempre realizar o acesso à aplicação a partir de um outro local, à hora que mais lhe aprouver, sem qualquer tipo de prejuízo, quer para o perito, quer para o próprio sistema.

- Múltiplas conclusões - Com o trabalho desenvolvido, apenas é possível, através do disparo de uma regra final, chegar a uma única conclusão. Deveria ser possível chegar a múltiplas conclusões pelo que, não tendo sido implementado nesta versão, considera-se como uma possibilidade de trabalho futuro.

  Aparentemente sem qualquer relação, a necessidade evidenciada no ponto anterior de criar um modo "‘supervisor"’ ou modo "‘perito"’, poderá levar um futuro desenvolvimento no sentido de criar as múltiplas conclusões aqui referidas já que, ao permitir este modo de funcionamento, também deverá ser possível inferir sobre vários cenários possíveis e respectivas conclusões.

- Negação de premissas - Nesta versão do trabalho, apenas é possível criar regras sem negação. Por exemplo, num cenário em que uma pergunta ao utilizador tem a forma:

  ▷ *Qual a cor? (Vermelho, Azul, Verde)*

  e as regras que são candidatas a disparo forem:

▷ *R1 - **SE** cor Vermelho, R2 - **SE** cor Azul, R3 **SE** cor Verde*

então, não é possível, caso seja necessário, ter uma regra do tipo:

▷ *Rx - **SE** cor **NOT** XXXXX*

Isso apenas seria possível se a pergunta inicial fosse:

▷ *Qual **NÃO É** a cor? (Vermelho, Azul, Verde)*

para assim permitir regras do tipo:

▷ *Rx - **SE NÃO É** cor XXXXX*

Assim sendo, aponta-se como uma limitação e eventual trabalho futuro a desenvolver para outras versões da aplicação a criação de um sistema que permita a negação de premissas para que o exemplo atrás indicado seja de possível implementação.

- Contemplar factos iniciais nas regras - A versão actual do sistema não permite a inclusão de factos inicias nas premissas das regras.
- Pesquisa de múltiplas soluções finais - Actualmente só é permitido executar o motor de inferência com uma solução final como objectivo. Seria uma evidente mais valia a inclusão da capacidade de pesquisa por múltiplas soluções.
- Metaconhecimento - ao ser adicionado metaconhecimento podemos aumentar o desempenho do motor de inferência nos cenários em que temos bases de conhecimento consideravelmente grandes.
- Regra mais relevante para iniciar o motor de inferência - desenvolver um algoritmo para obter uma regra que tenha uma maior probabilidade de sucesso de forma a minimizar iterações desnecessárias.
- Validação automática - quando as questões são apagadas, deve ser realizada a respectiva verificação de utilização dessas questões como condições de regras para prevenir possíveis inconsistências.
- Suporte a várias línguas - a base de conhecimento não foi projectada para implementar esta funcionalidade.

Bibliografia

Acquired Intelligence (2010). Acquire. http://www.aiinc.ca/, Junho de 2010. 25

Alexander Shapiro (2010). Tggraphlayout. http://sourceforge.net/projects/touchgraph/, Julho de 2010. 49, 79

Barnett, J. A. (1991). Calculating dempster-shafer plausibility. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **13**, 599–602. 16

Boris van Schooten (2005). Yprolog. http://www.vf.utwente.nl/~schooten/yprolog/, Junho de 2010. 37

Carlos Figueira Filho (2010). JEOPS - Integração entre Objetos e Regras de Produção em Java. http://www.cin.ufpe.br/~jeops/, Julho de 2010. 36

Co/AJRA, J. . (1997). The basics of expert (knowledge based) systems. 18

Dokas, I. M. (2005). Developing web sites for web based expert systems: A web engineering approach. In *In Proceedings of the Second International ICSC Symposium on Information Technologies in Environmental Engineering (Magdeburg*, pages 202–217. Shaker Verlag. 26

Drakopoulos, J. (1994). Probabilities, possibilities, and fuzzy sets. *Fuzzy Sets and Systems*, **75**, 1–15. 16

Elie Levy (2010). Zilonis. http://www.zilonis.org/index.php, Junho de 2010. 37

Ernest Friedman-Hill (2010). Jess. http://www.jessrules.com/, Julho de 2010. 25, 36

EXSYS (2010). Exsys corvid. http://www.exsys.com, Junho de 2010. 24

Filho, A. R. (2010). Mdss, medical diagnosis support system. LPA - Logic Programming Associates, Ltd. 30

Giarratano, C., J., Riley, and D., G. (1998). *Expert Systems - Principles and Programming, Thirds Edition: Principles and Programing*. Course Technology, 3ed edition. 6, 18

Hammurapi Group (2010). Hammurapi rules. http://www.hammurapi.com/dokuwiki/doku.php, Junho de 2010. 36

Heckerman, D. (1992). The certainty-factor model. 16

IBM (2010). Jlog. http://www-01.ibm.com/software/websphere/products/business-rule-management/, Junho de 2010. 36

Kozlenkov, Alex and Paschke (2010). Prova rule language. http://prova.ws/index.html, Junho de 2010. 37

Logic Programming Associaties Ltd (2010). Lpa webflex. http://www.lpa.co.uk/, Abril de 2010. 24

Lucas, P. (2001). Certainty-factor-like structures in bayesian belief networks. *Knowledge-Based Systems*, **14**, 327–335. 16

Maarten Menken (2010). Jclips. http://sourceforge.net/projects/jclips/, Julho de 2010. 25, 36

Mcarniel (2010). Jruleengine. http://jruleengine.sourceforge.net/index.html, Junho de 2010. 37

Merritt, D. (1989). Building expert systems in Prolog. 6, 56

Michael Giordano (2010). Webclips. http://clipsinterface.sourceforge.net/WebCLIPS/wchome.htm, 28 de Julho de 2003. 24

OpenL (2010). Openl tablets. http://openl-tablets.sourceforge.net/, Julho de 2010. 37

OpenRules, Inc. (2010). Open rules engine. http://openrules.com/index.htm, Julho de 2010. 37

Pinto, A. (1998). GISPSA - GIS problem solver adviser. 1, 20

RedHat (2010). Drools. http://www.jboss.org/drools/drools-expert.html, Junho de 2010. 36

Shortliffe, E. H. and Buchanan, B. G. (1990). A model of inexact reasoning in medicine. pages 259–275. 16

Wilson, N. (2000). Algorithms for dempster-shafer theory. In *Algorithms for Uncertainty and Defeasible Reasoning*, pages 421–475. Kluwer Academic Publishers. 16

Anexos

A

Código do motor de inferência

```
%%%%%%%%%%%%%%%%%%%%%%%%%%
% Início do motor de inferencia
%%%%%%%%%%%%%%%%%%%%%%%%%%
init_inference:- retractall(hist/2), consult('kbs.ypl').

top_goals(ControlObj, MinCF) :- top_goal(TopGoal), top(TopGoal, ControlObj, MinCF).

top(TopGoal, ControlObj, MinCF) :- retractall(hist/2),  set(len, 0),
getRuleMoreRelevant(TopGoal, Rule), tryToProove(Rule, ControlObj, MinCF).

tryToProove(Rule, ControlObj, MinCF):- rule(Rule, _, lhs(LHS), _),
proveListHyp(LHS, ControlObj, MinCF),get(len, LEN),println(LEN),
LEN1 is LEN + 1,assert(hist(LEN1,Rule)),println(LEN1),
set(len, LEN1),
(
  % check if rule has return control object
(var(ControlObj)) ->(
% no control object found, check if other rules needs to be fired
fireRules(ControlObj, MinCF)
);(
% control object found
true
)
).

proveListHyp([], ControlObj, MinCF).
proveListHyp([H|T], ControlObj, MinCF):-proveHyp(H, ControlObj, MinCF),
  ((var(ControlObj)) -> (proveListHyp(T, ControlObj, MinCF));(true)).


%fact already known
proveHyp(H, ControlObj, MinCF):- H = ev(X, Y), fact(X, Y,_ ),!.

% there is a question that can lead to a fact
proveHyp(H, ControlObj, MinCF):- H = ev(X, _),
not(fact(X, _, _)),question(X, _, _, _),
not(not_question(X)),ControlObj = ask(X), !.
```

```
%there is a conclusion that can lead to a question that can lead to a fact
proveHyp(H, ControlObj, MinCF):- H = ev(X, _), rule(Rule, _, lhs(LHS),
  rhs(ev(X,_), V)),
  tryToProove(Rule, ControlObj, MinCF), !.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ADD A FACT
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
av(Attr, Value, Confidence, ControlObj, MinCF) :- retractall(hist/2),set(len, 0),
saveFact(Attr, Value, Confidence, 1, CF_FINAL, _),!,
fireRules(ControlObj, MinCF), !.

nav(Attr):-assert(not_question(Attr)).

fireRules(ControlObj, MinCF):- rule(X, _, lhs(LHS), rhs(ev(Y, R), Conf)),
not(inf_rules_fired(X,_)), evaluate(LHS, ConfLhs, 1),
saveFinalValue(X, ConfLhs, Y, R, Conf, ConfRet, MinCF), !,
get(len, LEN), LEN1 is LEN + 1, assert(hist(LEN1,Rule)),
set(len, LEN1),
(
(var(ControlObj)) ->(
(top_goal(Y)) -> (
ControlObj = answerFound(Y, R, ConfRet)
);(
true,!, fireRules(ControlObj, MinCF)
));(
true
)
).


saveFinalValue(X, ConfLhs, Y, R, Conf, ConfRet, MinCF) :-
calculateConfidence(ConfLhs, Conf, ConfRet),
ConfRet > MinCF,
saveFact(Y, R, ConfRet, 2, CF_FINAL, X),
assert(inf_rules_fired(X, ConfRet)),!.


evaluate([], [], 1):-!, true.

evaluate([H|T], [FactConf|LConf] , RetCode):-
H = ev(X, Ans),fact(X, Ans, FactConf),
evaluate(T, LConf, RetCode),!.


calculateConfidence(ConfLhs, RuleCF, ResultCF):-min_in_list(ConfLhs, MinLHSCF),
ResultCF is MinLHSCF * RuleCF / 100.

min_in_list([Min],Min).
min_in_list([H,K|T],M) :- H <= K, min_in_list([H|T],M).
min_in_list([H,K|T],M) :- H > K,  min_in_list([K|T],M).

% 1 - fornecido, 2 - inferido
% we have a fact saved with the same conclusion already
```

```
saveFact(Attr, Value, Confidence, Mode, CF_Final, Rule):-
fact(Attr, Value, OldConfidence),
combine(OldConfidence, Confidence, CF_Final),
retract(fact(Attr, Value, OldConfidence)),
retract(inf_sys_fact(Attr, Value, OldConfidence, _)),
assert(fact(Attr, Value, CF_Final)),
assert(inf_sys_fact(Attr, Value, CF_Final, Mode)),
linkFactToRule(Rule, Attr, Value, Mode).

% we don't have any fact saved
saveFact(Attr, Value, Confidence, Mode, Confidence, Rule):-
assert(fact(Attr, Value, Confidence)),
assert(inf_sys_fact(Attr, Value, Confidence, Mode)), !,
linkFactToRule(Rule, Attr, Value, Mode).

linkFactToRule(_,_,_,1).
linkFactToRule(Rule,Attr,Value,2):-
inf_exp_how(Attr, Value, oldRules),
retract(inf_exp_how(Attr, Value, oldRules)),
assert(inf_exp_how(Attr, Value, [Rule|oldRules])).
linkFactToRule(Rule,Attr,Value,2):-
assert(inf_exp_how(Attr, Value, [Rule])),
println('saved line with information on explanation how'), listing.

combine(CF1, CF2, CF) :-  CF1 >= 0,  CF2 >= 0, MidPoint is 100 - CF1,
Calc is CF2 * MidPoint / 100, CF is CF1 + Calc.
combine(CF1, CF2, CF) :-  CF1 < 0,  CF2 < 0,  MidPoint is 100 - CF1,
Calc is CF2 * MidPoint / 100, CFAux is CF1 + Calc, CF is 0-CFAux.
combine(CF1, CF2, CF) :- (CF1 < 0; CF2 < 0), (CF1 > 0; CF2 > 0),
SumCFs is CF1 + CF2,  abs_minimum(CF1, CF2, MCF),
Rest is 100 - MCF,CF is 100 * SumCFs / Rest.


abs_minimum(A,B,X) :- absolute(A, AA), absolute(B, BB),minimum(AA,BB,X).

absolute(X, X) :-X >= 0.
absolute(X, Y) :-X < 0,Y is 0-X.

minimum(X,Y,X) :-X <= Y,!.
minimum(X,Y,Y) :-Y <= X.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Rule verification
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
verification(L):-retractall(validation_fired/1),
retractall(verification_error/4),set(nextid, 0),
findall(ID,verification_rule_1(ID, _, _, _),L).

verification_rule_1(ID, RuleInError, RuleRelated, Result):-
rule(A, _, lhs(CA), RA),rule(B, _, lhs(CB), RB),
not( B = A),not(validation_fired(B)),
checkConditionsSimilar(CA, CB, FullMatch),
checkResultSimilar(FullMatch, RA, RB, Result),
assert(validation_fired(B)),
```

```
get(nextid, CURR_ID),ID is CURR_ID + 1,
set(nextid, ID),
assert(verification_error(ID, B, A, Result)),
RuleInError = B, RuleRelated = A.

checkConditionsSimilar(CA, CB, FullMatch):-checkConditionsSimilar_1(CA,CB),
lent(CA, X),  lent(CB, Y), testCheck(X, Y, FullMatch).

testCheck(X, X, 1).
testCheck(X, Y, 0):- not(X = Y).

checkConditionsSimilar_1([], L).
checkConditionsSimilar_1([H|R], L):- member(H, L), checkConditionsSimilar_1(R, L).


checkResultSimilar(1, rhs(ev(Atom, ValueA), CFA), rhs(ev(Atom, ValueB), CFB), 2):-
not(ValueA = ValueB).
checkResultSimilar(1, rhs(ev(Atom, ValueA), CFA), rhs(ev(Atom, ValueA), CFB), 2).
checkResultSimilar(0, rhs(ev(Atom, Value), CFA), rhs(ev(Atom, Value), CFB), 1):-
  CFB <= CFA.


lent([], 0).
lent([_|R], SizeT):- lent(R, Size),SizeT is Size + 1.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Rule verification
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
retrieve_rules(L):-
findall(X, rule(X, _, _,_), L).

retrieve_questions(L):-
findall(X, question(X, _,_,_), L).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Information predicates
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
inf_sys(L_Rules, L_Inf_Facts, L_Know_Facts, L_Init_Facts):-
i_sys_rules(L_Rules),i_sys_inf_fact(L_Inf_Facts),
i_sys_know_fact(L_Know_Facts), i_sys_init_fact(L_Init_Facts).

i_sys_rules(L):-
findall(r(Rule, Conf), inf_rules_fired(Rule, Conf), L), member(C, L).
i_sys_rules([]).

i_sys_inf_fact(L):-
findall(f(Attr, Value, Conf), inf_sys_fact(Attr, Value, Conf, 2), L), member(C, L).
i_sys_inf_fact([]).

i_sys_know_fact(L):-
findall(f(Attr, Value, Conf), inf_sys_fact(Attr, Value, Conf, 1), L), member(C, L).
i_sys_know_fact([]).
```

```
i_sys_init_fact(L):-
findall(f(Attr, Value, Conf), condition_init(Attr, Value, Conf), L), member(C, L).
i_sys_init_fact([]).

condition_init(Attr, Value, Conf):-fact(Attr, Value, Conf),
  not(inf_sys_fact(Attr, _, _, _)).

%%%%%%%%%%%%%%%%%%%%%%%%%%%
% explanations
%%%%%%%%%%%%%%%%%%%%%%%%%%%

why(L):-findall(r(O,X), hist(O,X), L).

how(Attr, Value, Result):-  inf_exp_how(Attr, Value, Rules),
explainHow(Rules, Explanation),!,append(Rules, Explanation, Result).

explainHow([], []).
explainHow([H|T], Result):- innerExplain(H, ExplanationH),!,
    explainHow(T, ExplanationRules), append(ExplanationH, ExplanationRules, Result).


innerExplain(Rule, Explanation):- rule(Rule, _, lhs(LHS), _),!,
explainLHS(LHS, Explanation).

explainLHS([], []).
explainLHS([ev(Attr, Value)|T], ExplanationA):-
not(inf_exp_how(Attr, Value, _)),!,
fact(Attr, Value, _), print('initial fact'),!,
explainLHS(T, ExplanationA), !.

explainLHS([ev(Attr, Value)|T], ResultA):-
how(Attr, Value, ExplanationA),!,
explainLHS(T, Explanation), !,
append(ExplanationA, Explanation, ResultA).


append([], L, L).
append(L, [], L).
append([H|T], M, [H|N]):-append(L, M, N).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Map methods
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

buildValidationReport(Id, RErr, RRel, ErrType):-
  verification_error(Id, RErr, RRel, ErrType).


buildQuestionObject(ask(X), X, Question, LOut, Multi):-
question(X, Question, L, Multi), flatten(L, LOut).

buildQuestionObject_by_code(X, X, Question, LOut, Multi):-
question(X, Question, L, Multi), flatten(L, LOut).
```

```
buildSolutionObject(answerFound(Y, R, Found), Y, R, Found).

buildRuleObject(Rule, Descr, LHS, RHS, CF):-
rule(Rule, Descr, lhs(I_LHS), rhs(I_RHS, CF)),
lhs_flatten(I_LHS, LHS),
rhs_flatten(I_RHS, RHS).


flatten(L, LO):- iflatten(L, LO).
iflatten([], []).
iflatten([H|R], [A,D|L]):- H = answer(A, D), iflatten(R, L).

lhs_flatten(L, LO):- ilhs_iflatten(L, LO).

ilhs_iflatten([], []).
ilhs_iflatten([H|R], [[A,D,C,B]|L]):-H = ev(A, D),question(A, C, LL, _),
    !,findMember(LL,D,B),!,ilhs_iflatten(R, L), !.

ilhs_iflatten([H|R], [[A,D,A,D]|L]):-H = ev(A, D),not(question(A, _, _, _)),
ilhs_iflatten(R, L), !.

rhs_flatten(ev(A,B), [A,B]).

findMember([], D, D):-!,.
findMember([H|T], D, B):-H = answer(D, B), !.
findMember([H|T], D, B):-findMember(T, D, B).

% ######################################
% ###    attempt to  Get more relevant rule     ###
% ######################################
getRuleMoreRelevant(TopGoal, Rule):-
rule(Rule,_, _, rhs(ev(P, V), X)), top_goal(P).

% find the rules that contains a direct solution
%findall(X, rule(X, lhs(LLHS), rhs(ev(TopGoal, _),_)), LX),
%debugLine(trc1,'>'),debugList(LX),
% For the main rules apply the algorithm
%fillMainRules(LX, LOUT),
%bubble_sort(LOUT, LREALLYOUT),
%LREALLYOUT = [ score_table(Rule, _, _, _) | _ ],
%debugLine(trc1,'>'),debugList(LREALLYOUT)
%.

fillMainRules([], []):-!.
fillMainRules([HIN|RIN], [HOUT|LOUT]):- calculateCurrentRule(HIN, HOUT),
fillMainRules(RIN, LOUT).

calculateCurrentRule(HIN, HOUT):-rule(HIN, _, lhs(LHS), rhs(_, Confidence)),
calculateScoreBasedOnLHS(LHS, 0, Score),
OrderIndice is Score * Confidence,
HOUT = score_table(HIN, Score, Confidence, OrderIndice), !.

calculateScoreBasedOnLHS([], ScoreResult, ScoreResult):-!.
```

```
calculateScoreBasedOnLHS([HLHS|RLHS], Score, ScoreResult):-
calculateScoreBasedOnLHSCurrent(HLHS, Score, ScoreAfter),
calculateScoreBasedOnLHS(RLHS, ScoreAfter, ScoreResult), !.

calculateScoreBasedOnLHSCurrent(ev(X, _), Score, ScoreAfter):-
question(X, _, _, _), not(fact(X, _, _)),
ScoreAfter is Score + 1, !.
calculateScoreBasedOnLHSCurrent(ev(X, _), Score, Score):-!.


% ##############################
% ###      helper methods     ###
% ##############################
bubble_sort(List,Sorted):-b_sort(List,[],Sorted).
b_sort([],Acc,Acc).
b_sort([H|T],Acc,Sorted):-bubble(H,T,NT,Max),b_sort(NT,[Max|Acc],Sorted).

bubble(X,[],[],X).
bubble(X,[Y|T],[Y|NT],Max):- X = score_table(_, _, _, C),
Y = score_table(_, _, _, D), C <= D, bubble(X,T,NT,Max).
bubble(X,[Y|T],[X|NT],Max):- X = score_table(_, _, _, C),
Y = score_table(_, _, _, D), C > D,bubble(Y,T,NT,Max).

bubble_sort2(List,Sorted):-b_sort2(List,[],Sorted).
b_sort2([],Acc,Acc).
b_sort2([H|T],Acc,Sorted):-bubble2(H,T,NT,Max),b_sort2(NT,[Max|Acc],Sorted).

bubble2(X,[],[],X).
bubble2(X,[Y|T],[Y|NT],Max):- X = r(D, _), Y = r(C, _), C <= D, bubble(X,T,NT,Max).
bubble2(X,[Y|T],[X|NT],Max):- X = r(D, _), Y = r(C, _), C > D,bubble(Y,T,NT,Max).

% Allows to get a Bag of all the X that makes Goal evaluate to true
% Bag can contain duplicated X
findall(X,Goal,Bag) :-   post_it(X,Goal), gather([],Bag).

% Aux
post_it(X,Goal) :- call(Goal), asserta(data999(X)), fail.
post_it(_,_).

gather(B,Bag) :-  data999(X), retract(data999(X)), gather([X|B],Bag), !.
gather(S,S).

member(X,[X|_]):-!.
member(X,[_|Y]) :- member(X,Y).
```

B

Base de conhecimento do protótipo

```
question(turn_over, 'O carro liga?',
 [ answer(yes, 'Sim'), answer(no, 'Não')],0).

question(lights_weak, 'As luzes estão fracas?',
 [ answer(yes, 'Sim'), answer(no, 'Não')],0).

question(radio_weak, 'O rádio está fraco?',
 [ answer(yes, 'Sim'), answer(no, 'Não')],0).

question(smell_gas, 'Cheira a gasolina?',
 [ answer(yes, 'Sim'), answer(no, 'Não')],0).

question(gas_gauge, 'O que marca o ponteiro do depósito da gasolina?',
 [ answer(empty, 'Vazio'), answer(high, 'Suficiente'), answer(low, 'Reserva')],0).

rule(rule_1, 'Rule 1',lhs([ev(turn_over, no), ev(battery_bad, yes)]),
  rhs(ev(problem, battery), 100)).

rule(rule_2, 'Rule 2',lhs([ev(lights_weak, yes)]),
 rhs(ev(battery_bad, yes), 50)).

rule(rule_3, 'Rule 3',lhs([ev(radio_weak, yes)]),
 rhs(ev(battery_bad, yes), 50)).

rule(rule_4, 'Rule 4',lhs([ev(smell_gas, yes), ev(turn_over, yes)]),
  rhs(ev(problem, flooded), 80)).

rule(rule_5, 'Rule 5',lhs([ev(turn_over, yes), ev(gas_gauge, empty)]),
  rhs(ev(problem, out_of_gas), 90)).

rule(rule_6, 'Rule 6',lhs([ev(turn_over, yes), ev(gas_gauge, low)]),
  rhs(ev(problem, out_of_gas), 30)).

rule(rule_7, 'Rule 7',lhs([ev(gas_gauge, low),ev(turn_over, yes)]),
  rhs(ev(problem, out_of_gas), 30)).
```

```
rule(rule_8, 'Rule 8',
 lhs([ev(turn_over, yes), ev(gas_gauge, low), ev(smell_gas, no)]),
 rhs(ev(problem, out_of_gas), 30)).

top_goal(problem).
```

C

Base de conhecimento GISPSA

```
question(qs_tema_ger, 'Qual o tipo de Problema?',
 [ answer(c_ger_map, 'Mapas Configuráveis'), answer(c_ger_inq, 'Inquéritos e Listagens'),
  answer(c_ger_mov, 'CGMOVIG'), answer(c_ger_pla, 'CGDCGER'),
answer(c_ger_acm, 'CGVACUM/CGTACUM')],0).

question(qs_plareg, 'A Conta tem regra de construcção associada ?',
 [ answer(c_nao, 'Não'), answer(c_sim, 'Sim')],0).

question(qs_platipr, 'Existe Conta Pai do Tipo R no Plano',
 [ answer(c_nao, 'Não'), answer(c_sim, 'Sim')],0).

question(qs_plaent, 'Conta aberta por entidades',
 [ answer(c_nao, 'Não'), answer(c_sim, 'Sim')],0).

question(qs_platipa, 'Conta do Tipo A',
 [ answer(c_nao, 'Não'), answer(c_sim, 'Sim')],0).

question(qs_plalim, 'Entidade encontra-se nos limites',
 [ answer(c_nao, 'Não'), answer(c_sim, 'Sim')],0).

question(qs_plaexis, 'Entidade Existente?',
 [ answer(c_nao, 'Não'), answer(c_sim, 'Sim')],0).

question(qs_plamix, 'Contas Valor entre Contas Movimento?',
 [ answer(c_nao, 'Não'), answer(c_sim, 'Sim')],0).

question(qs_acumul, 'Qual o problema de acumulados?',
 [ answer(c_acm_avm, 'CGVACUM <> CGMOVIG'),
   answer(c_acm_si, 'Saldos Iniciais'),
   answer(c_acm_cna, 'Contas Nao Acumulam'),
   answer(c_acm_avat, 'CGVACUM <> CGTACUM')],0).

question(qs_incmov, 'Inconsistências no CGMOVIG?',
 [ answer(c_nao, 'Não'), answer(c_sim, 'Sim')],0).
```

```
question(qs_incpla, 'Inconsistências no CGDCGER?',
 [ answer(c_nao, 'Não'), answer(c_sim, 'Sim')],0).

question(qs_acmsia, 'Saldos Iniciais Alterados?',
 [ answer(c_nao, 'Não'), answer(c_sim, 'Sim')],0).

question(qs_movi, 'Qual o problema dos movimentos?',
 [ answer(c_mov_lan, 'Lançamentos nao balanceados'),
   answer(c_mov_con, 'Problema de Contas'),
   answer(c_mov_cgca, 'Movimentos CG <> CA')],0).

question(qs_movvdif, 'Valores CG > CA?',
 [ answer(c_nao, 'Não'), answer(c_sim, 'Sim')],0).

question(qs_movlgca, 'C.C. Ligados a Analitica?',
 [ answer(c_nao, 'Não'), answer(c_sim, 'Sim')],0).

question(qs_movlg100, 'Todas as Ligacoes a 100%?',
 [ answer(c_nao, 'Não'), answer(c_sim, 'Sim')],0).

question(qs_movm100, 'Diferencas Correspondem a C.C. com Mais de 100%?',
 [ answer(c_nao, 'Não'), answer(c_sim, 'Sim')],0).

question(qs_movcna, 'Lancamento em Contas nao Existentes?',
 [ answer(c_nao, 'Não'), answer(c_sim, 'Sim')],0).

question(qs_movci, 'Lancamento em Contas Invalidas?',
 [ answer(c_nao, 'Não'), answer(c_sim, 'Sim')],0).

question(qs_movcdf, 'Contas correctas?',
 [ answer(c_nao, 'Não'), answer(c_sim, 'Sim')],0).

question(qs_inquer, 'Inquérito ou listagem?',
 [ answer(c_inq_bal, 'Balancete'), answer(c_inq_dia, 'Extracto/Diario')],0).

question(qs_inqsal, 'Problema de saldos?',
 [ answer(c_nao, 'Não'), answer(c_sim, 'Sim')],0).

question(qs_mapa, 'Problema dos mapas?',
 [ answer(c_map_imp, 'Erro na Impressao'), answer(c_map_cal, 'Erro no Calculo'),
   answer(c_map_mic, 'Erro na Transferencia P/ Micro')],0).

question(qs_mapcol0, 'A ultima linha calculada tem a coluna 0 configurada?',
 [ answer(c_nao, 'Não'), answer(c_sim, 'Sim')],0).

question(qs_mapfunc,
 'A ultima linha calculada apresenta colunas com funcoes invalidas ou sintaxe errada?',
 [ answer(c_nao, 'Não'), answer(c_sim, 'Sim')],0).

question(qs_maplin0, 'A Linha 000 esta configurada?',
 [ answer(c_nao, 'Não'), answer(c_sim, 'Sim')],0).

question(qs_maplin9, 'A Linha 999 esta configurada?',
 [ answer(c_nao, 'Não'), answer(c_sim, 'Sim')],0).
```

```
question(qs_map2alg, 'Todos os Valores da Linha 999 tem 2 Algarismos?',
 [ answer(c_nao, 'Não'), answer(c_sim, 'Sim')],0).

question(qs_map198, 'A Soma dos Valores da Linha 999 Ultrapassa 198?',
 [ answer(c_nao, 'Não'), answer(c_sim, 'Sim')],0).

question(qs_mapsys, 'O Sistema Permite a Utilizacao de Folders?',
 [ answer(c_nao, 'Não'), answer(c_sim, 'Sim')],0).

question(qs_mapuser, 'O Utilizador Tem Acesso a Folders?',
 [ answer(c_nao, 'Não'), answer(c_sim, 'Sim')],0).

question(qs_mapflr, 'O Folder Existe?',
 [ answer(c_nao, 'Não'), answer(c_sim, 'Sim')],0).

rule(r_pla_1, r_pla_1,lhs([ev(qs_tema_ger, c_ger_pla), ev(qs_plareg, c_sim)]),
 rhs(ev(dominio, d_pla_reg), 100)
).
rule(r_pla_2, r_pla_2,lhs([ev(qs_tema_ger, c_ger_pla), ev(qs_plaent, c_sim)]),
 rhs(ev(dominio, d_pla_ent), 100)
).
rule(r_pla_3, r_pla_3,lhs([ev(qs_tema_ger, c_ger_pla), ev(qs_plamix, c_sim)]),
 rhs(ev(correccao, cr_pla_mix), 100)
).
rule(r_pla_11, r_pla_11,lhs([ev(dominio, d_pla_reg), ev(qs_platipr, c_nao)]),
 rhs(ev(correccao, cr_pla_tipr), 100)
).
rule(r_pla_21, r_pla_21,lhs([ev(dominio, d_pla_ent), ev(qs_platipa, c_nao)]),
 rhs(ev(correccao, cr_pla_tipa), 100)
).
rule(r_pla_22, r_pla_22,lhs([ev(dominio, d_pla_ent), ev(qs_plalim, c_nao)]),
 rhs(ev(correccao, cr_pla_lim), 100)
).
rule(r_pla_23, r_pla_23,lhs([ev(dominio, d_pla_ent), ev(qs_plaexis, c_nao)]),
 rhs(ev(correccao, cr_pla_exis), 100)
).
rule(r_acm_1, r_acm_1,
 lhs([ev(qs_tema_ger, c_ger_acm), ev(qs_acumul, c_acm_avm), ev(qs_incmov, c_sim)]),
 rhs(ev(correccao, cr_acm_avat), 100)
).
rule(r_acm_2, r_acm_2,
 lhs([ev(qs_tema_ger, c_ger_acm), ev(qs_acumul, c_acm_avm), ev(qs_incmov, c_nao)]),
 rhs(ev(correccao, cr_acm_avat), 100)
).
rule(r_acm_3, r_acm_3,
 lhs([ev(qs_tema_ger, c_ger_acm), ev(qs_acumul, c_acm_avat), ev(qs_incpla, c_sim)]),
 rhs(ev(correccao, cr_acm_at), 100)
).
rule(r_acm_4, r_acm_4,
 lhs([ev(qs_tema_ger, c_ger_acm), ev(qs_acumul, c_acm_avat), ev(qs_incpla, c_nao)]),
 rhs(ev(correccao, cr_acm_at), 100)
).
rule(r_acm_5, r_acm_5,
```

```
 lhs([ev(qs_tema_ger, c_ger_acm), ev(qs_acumul, c_acm_cna), ev(qs_incpla, c_sim)]),
 rhs(ev(correccao, cr_acm_avat), 100)
).
rule(r_acm_6, r_acm_6,
 lhs([ev(qs_tema_ger, c_ger_acm), ev(qs_acumul, c_acm_cna), ev(qs_incpla, c_nao)]),
 rhs(ev(correccao, cr_acm_avat), 100)
).
rule(r_acm_7, r_acm_7,
 lhs([ev(qs_tema_ger, c_ger_acm), ev(qs_acumul, c_acm_si), ev(qs_acmsia, c_nao)]),
 rhs(ev(correccao, cr_acm_si), 100)
).
rule(r_mov_1, r_mov_1,lhs([ev(qs_tema_ger, c_ger_mov), ev(qs_movi, c_mov_lan)]),
 rhs(ev(correccao, cr_mov_inc), 100)
).
rule(r_mov_2, r_mov_2,lhs([ev(qs_tema_ger, c_ger_mov), ev(qs_movi, c_mov_cgca)]),
 rhs(ev(dominio, d_mov_cgca), 100)
).
rule(r_mov_21, r_mov_21,lhs([ev(dominio, d_mov_cgca), ev(qs_movvdif, c_sim)]),
 rhs(ev(subdominio, d_mov_cgca_1), 100)
).
rule(r_mov_22, r_mov_22,lhs([ev(dominio, d_mov_cgca), ev(qs_movm100, c_sim)]),
 rhs(ev(correccao, cr_mov_inc_cgca), 100)
).
rule(r_mov_23, r_mov_23,lhs([ev(dominio, d_mov_cgca)]),
 rhs(ev(correccao, cr_mov_inc), 100)
).
rule(r_mov_3, r_mov_3,lhs([ev(qs_tema_ger, c_ger_mov), ev(qs_movi, c_mov_con)]),
 rhs(ev(dominio, d_mov_con), 100)
).
rule(r_mov_31, r_mov_31,lhs([ev(dominio, d_mov_con), ev(qs_movcna, c_sim)]),
 rhs(ev(correccao, cr_mov_inc), 100)
).
rule(r_mov_32, r_mov_32,lhs([ev(dominio, d_mov_con), ev(qs_movci, c_nao)]),
 rhs(ev(correccao, cr_mov_inc_soft), 100)
).
rule(r_mov_33, r_mov_33,lhs([ev(dominio, d_mov_con), ev(qs_movcdf, c_nao)]),
 rhs(ev(correccao, cr_mov_inc), 100)
).
rule(r_mov_34, r_mov_34,lhs([ev(dominio, d_mov_con)]),
 rhs(ev(correccao, cr_mov_inc), 100)
).
rule(r_mov_211, r_mov_211,lhs([ev(subdominio, d_mov_cgca_1), ev(qs_movlgca, c_nao)]),
 rhs(ev(correccao, cr_mov_inc_cgca), 100)
).
rule(r_mov_212, r_mov_212,lhs([ev(subdominio, d_mov_cgca_1), ev(qs_movlg100, c_sim)]),
 rhs(ev(correccao, cr_mov_inc), 100)
).
rule(r_mov_213, r_mov_213,lhs([ev(subdominio, d_mov_cgca_1)]),
 rhs(ev(correccao, cr_mov_inc_cgca), 100)
).
rule(r_inq_1, r_inq_1,lhs([ev(qs_tema_ger, c_ger_inq), ev(qs_inquer, c_inq_bal)]),
 rhs(ev(correccao, cr_inq_bal), 100)
).
rule(r_inq_2, r_inq_2,lhs([ev(qs_tema_ger, c_ger_inq), ev(qs_inqsal, c_sim)]),
```

```
 rhs(ev(correccao, cr_inq_dia), 100)
).
rule(r_map_1, r_map_1,lhs([ev(qs_tema_ger, c_ger_map), ev(qs_mapa, c_map_cal)]),
 rhs(ev(dominio, d_map_cal), 100)
).
rule(r_map_2, r_map_2,lhs([ev(qs_tema_ger, c_ger_map), ev(qs_mapa, c_map_imp)]),
 rhs(ev(dominio, d_map_imp), 100)
).
rule(r_map_3, r_map_3,lhs([ev(qs_tema_ger, c_ger_map), ev(qs_mapa, c_map_mic)]),
 rhs(ev(dominio, d_map_mic), 100)
).
rule(r_map_11, r_map_11,lhs([ev(dominio, d_map_cal), ev(qs_mapcol0, c_sim)]),
 rhs(ev(correccao, cr_map_cal_col0), 100)
).
rule(r_map_12, r_map_12,lhs([ev(dominio, d_map_cal), ev(qs_mapfunc, c_sim)]),
 rhs(ev(correccao, cr_map_cal_func), 100)
).
rule(r_map_13, r_map_13,lhs([ev(dominio, d_map_cal)]),
 rhs(ev(correccao, cr_map_soft), 100)
).
rule(r_map_21, r_map_21,lhs([ev(dominio, d_map_imp), ev(qs_maplin0, c_nao)]),
 rhs(ev(correccao, cr_map_imp_lin0), 100)
).
rule(r_map_22, r_map_22,lhs([ev(dominio, d_map_imp), ev(qs_maplin9, c_nao)]),
 rhs(ev(correccao, cr_map_imp_lin9), 100)
).
rule(r_map_23, r_map_23,lhs([ev(dominio, d_map_imp), ev(qs_map2alg, c_nao)]),
 rhs(ev(correccao, cr_map_imp_2alg), 100)
).
rule(r_map_24, r_map_24,lhs([ev(dominio, d_map_imp), ev(qs_map198, c_nao)]),
 rhs(ev(correccao, cr_map_imp_198), 100)
).
rule(r_map_25, r_map_25,lhs([ev(dominio, d_map_imp)]),
 rhs(ev(correccao, cr_map_soft), 100)
).
rule(r_map_31, r_map_31,lhs([ev(dominio, d_map_mic), ev(qs_mapsys, c_nao)]),
 rhs(ev(correccao, cr_map_sys), 100)
).
rule(r_map_32, r_map_31,lhs([ev(dominio, d_map_mic), ev(qs_mapuser, c_nao)]),
 rhs(ev(correccao, cr_map_user), 100)
).
rule(r_map_33, r_map_33,lhs([ev(dominio, d_map_mic), ev(qs_mapflr, c_nao)]),
 rhs(ev(correccao, cr_map_mic_flr), 100)
).
rule(r_map_34, r_map_34,lhs([ev(dominio, d_map_mic)]),
 rhs(ev(correccao, cr_map_soft), 100)
).

top_goal(correccao).
```