Inteligência Artificial

Metodologia CommonKADS
Utilizando um formalismo para modelar conhecimento

Prof. Paulo Martins Engel

História dos sistemas de conhecimento

- Máquinas de busca de propósito geral (1965)
- Primeira geração de sistemas baseados em regras (1975 – MYCIN, XCON)
- Emergência de métodos estruturados (1985 – KADS, Knowledge Acquisition and Documentation Structuring)
- Metodologias maduras (1995 – CommonKADS)

Princípios de CommonKADS

- **Princípio da Modelagem**
- Engenharia de conhecimento não é apenas extração de conhecimento do especialista, mas consiste na construção de modelos de diversos aspectos do conhecimento humano

Princípios de CommonKADS

- **Princípio do nível de conhecimento**
- Ao modelar conhecimento, primeiro deve-se concentrar na estrutura conceitual do conhecimento e deixar os detalhes de implementação para depois.

Princípios de CommonKADS

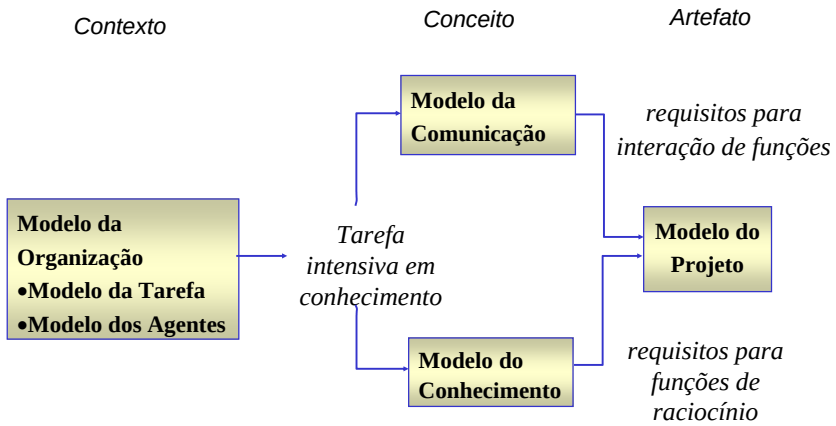
- **O conhecimento tem uma estrutura interna estável que é analisável, distinguindo-se *tipos* e *papéis* específicos do conhecimento**
- O conhecimento tem uma estrutura interna que reutiliza padrões similares de conhecimento (*tipos*).
- Os tipos de conhecimento assumem papéis diferentes, limitados, na solução de problemas (*limitação de papéis*).

Primitivas do Modelo

- Conhecimento
- Objetivos
- Ações

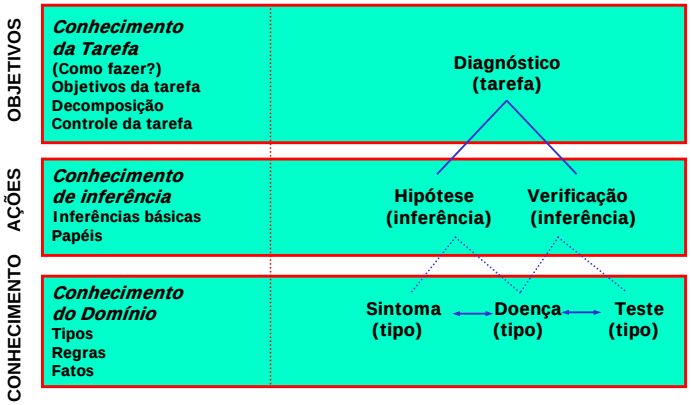
- **Modelo do Domínio**
- **Modelo da Tarefa**
- **Métodos de Solução de Problemas** (PSM)

O Conjunto de Modelos *CommonKADS*



Modelo do Conhecimento

- Representando o conhecimento em um formalismo
- Categorias de conhecimento



Conhecimento do Domínio
Ontologias

- Especificação formal e explícita de um conjunto de conceitos compartilhados
  - Explícito: conceitos e restrições previamente definidos
  - Formal: processável por computador
  - Compartilhada: descreve um conhecimento consensual, que é aceito por um grupo.
- Inclui conceitos, relações, regras e todos os tipos estáticos.

Ontologia

(i) Um *vocabulário* de conceitos, ou termos do domínio;

(ii) A *tipologia* do domínio, que define tipos de dados e restrições de valores que os termos devem respeitar;

(iii) As *relações* entre conceitos, que formam as *taxonomias* e *partonomias* daquele domínio, ou outras associações entre conceitos.

Construtores do Esquema do Domínio

- Conceitos: objetos ou entidades do domínio
  - Definidos por seus atributos e tipos de valores
- Relações : classificação, especialização, agregação, conjunto
- Tipo-regra : relações de dependências entre conceitos do domínio ou entre expressões de domínio

Conceitos

- Conceitos do domínio: objetos ou entidades
- os atributos ou propriedades desses objetos;
- as restrições que definem os conceitos e distinguem a realidade deste domínio dos demais
- (Descrevem o que são as coisas do domínio)

Conceitos

marcador-combustível
valor: valor-marcador

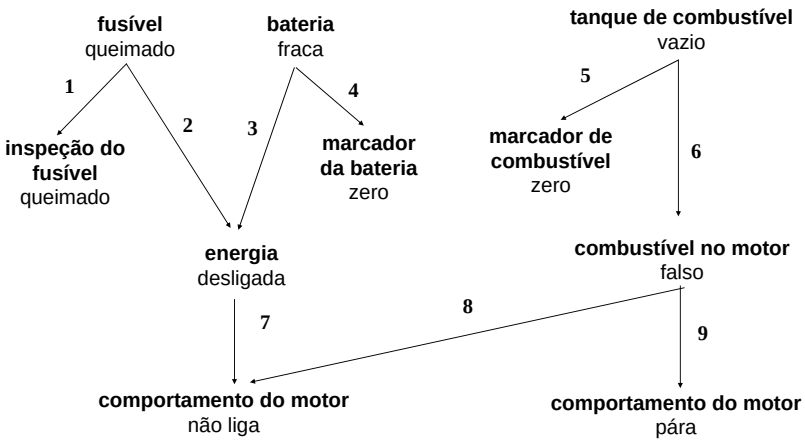
CONCEPT marcador-combustível;
ATTRIBUTES
valor: valor-marcador;
END-CONCEPT marcador-combustível;

tanque-combustível
status: {cheio, reserva, vazio}

CONCEPT tanque-combustível;
ATTRIBUTES
status: {cheio, reserva, vazio};
END-CONCEPT tanque-combustível;

VALUE-TYPE valor-marcador;
VALUE-LIST: {zero, baixo, normal};
TYPE: ORDINAL;
END-VALUE-TYPE valor-marcador;

Exemplo de elementos de conhecimento do domínio do diagnóstico de um carro



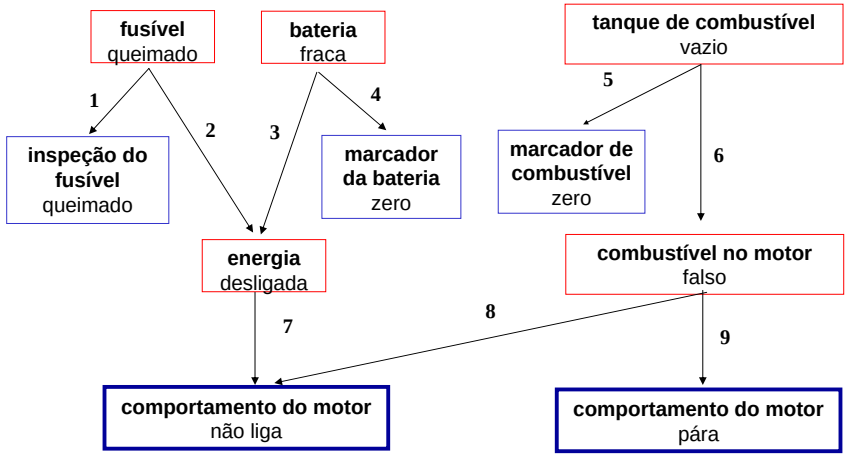
Relações do Domínio

Principalmente

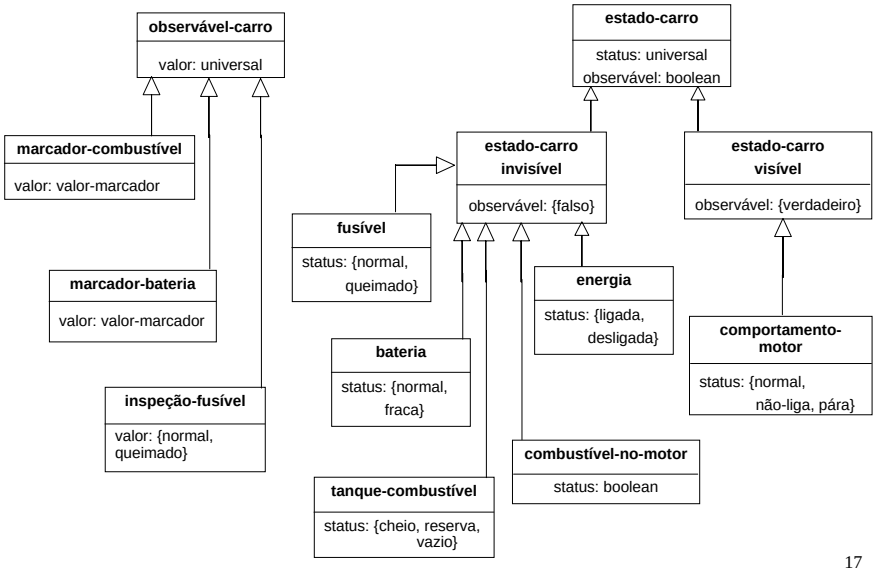
- Classificação
- Especialização
- Agregação
- Associação de conjunto

Tipos de conceitos

estado invisível | estado visível | observável



Relações de subtipo entre conceitos



Bases de Conhecimento

- Instâncias dos conceitos do domínio que descrevem a aplicação
- **Não** são instâncias do usuário

```
INSTANCE tanque-combustível
     status: vazio
  END-INSTANCE tanque-combustível
```

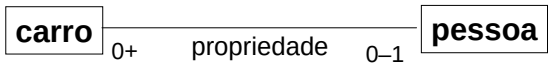

Termos de Domínio

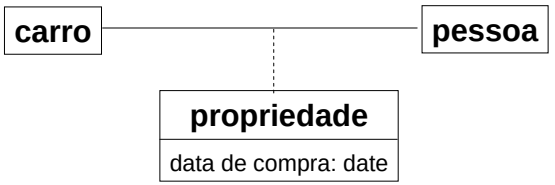
- Declarações sobre os conceitos do domínio ou qualquer combinação lógica dessas declarações

```
tanque-combustível.status = vazio => combustível-
  no-motor.status = falso
bateria.status = fraca => energia.status =
  desligada
farol.status = não-funciona E ignição.status =
  não-funciona E radio.status = não-funciona
```


Relações

- Declarações sobre relações entre conceitos podem ser binárias ou múltiplas





Dependências entre estados do carro

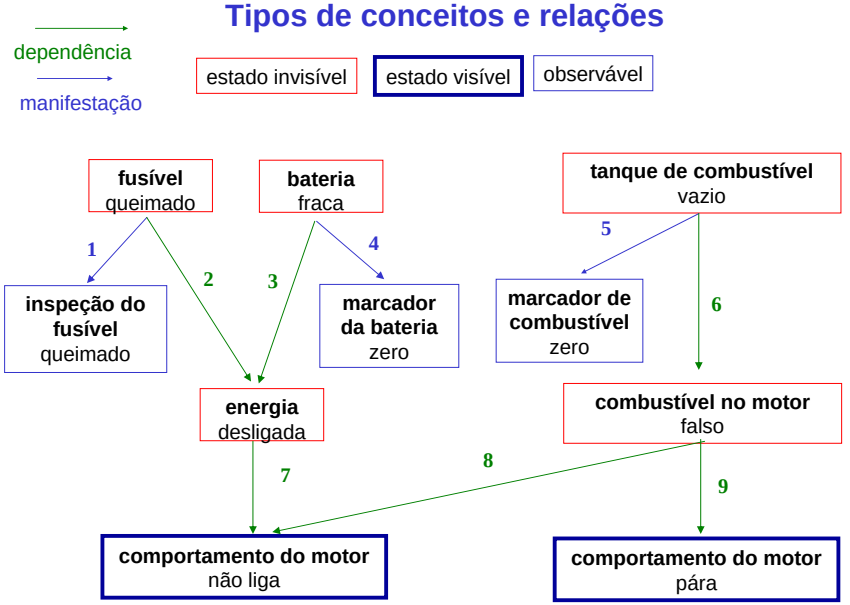
```
tanque-combustível.status = vazio =>
     combustível-no-motor.status = falso
bateria.status =fraca =>
     energia.status = desligada
```

São relações entre *expressões* sobre conceitos

Tipo-regra

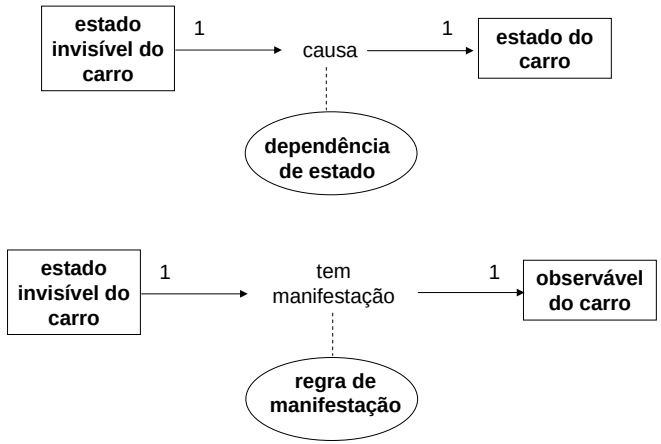
- Descreve dependências entre *expressões* do domínio.
- Podem ser regras abstratas ou suas instâncias
- No exemplo, são definidos dois tipos de regra:
  - Regras de *dependência* entre estados
  - Regras de *manifestação* de estado invisível

Tipos de conceitos e relações



Regras

- Regras de dois tipos: *dependência* e *manifestação*



Tipo-regra

- Tipo de regra de dependência de estados

```
RULE-TYPE dependência-estado;
  ANTECEDENTE: estado-invisível;
     CARDINALIDADE: 1;
  CONSEQUENTE: estado-carro;
     CARDINALIDADE: 1;
  SIMBOLO-CONEXÃO: causa
END RULE-TYPE dependência-estado;
```


Dependência de estados

```
fusível.status = queimado CAUSA
     energia.status = desligada;
bateria.status = fraca CAUSA
     energia.status = desligada;
energia.status = desligada CAUSA
     comportamento-motor.status = não-liga;
tanque-combustível.status = vazio CAUSA
     combustível-no-motor.status = falso;
combustível-no-motor.status = falso CAUSA
     comportamento-motor.status = não-liga;
combustível-no-motor.status = falso CAUSA
     comportamento-motor.status = pára;
```


Outro tipo de regra

- Regras representam manifestações típicas dos estados internos.

```
RULE-TYPE regra-manifestação;
  DESCRIÇÃO: “Representa a relação entre um
  estado interno e seu comportamento externo
  através de um valor observável”;
  ANTECEDENTE: estado-invisível;
  CONSEQUENTE: observável-carro;
     CARDINALIDADE: 1;
  SIMBOLO-CONEXÃO: tem-manifestação;
END RULE-TYPE regra-manifestação;
```


Regras de manifestação de estados

```
fusível.status = queimado TEM-MANIFESTAÇÃO
   inspeção-fusível.status = queimado;
bateria.status = fraca TEM-MANIFESTAÇÃO
   marcador-bateria.valor = zero;
tanque-combustível.status = vazio TEM-
   MANIFESTAÇÃO marcador-combustível.valor =
  zero;
```

Modelo da Tarefa

- Define a relação entre os conceitos do domínio necessários à aplicação e os métodos de solução de problema abstratos necessários para obter a solução.
- Realiza a *instanciação* dos métodos de solução de problemas

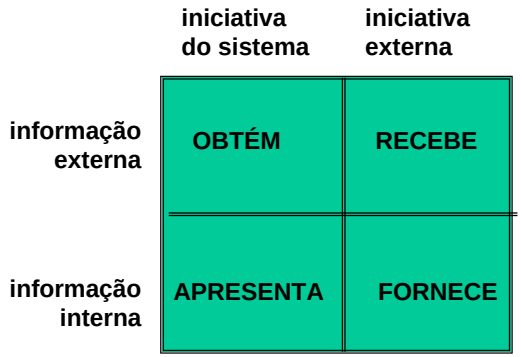
Modelo da Tarefa - Definido por ...

- Objetivo
- Papéis de entrada e saída
- Corpo da tarefa
  - sub-objetivos
  - sub-tarefas
  - estrutura de controle: em que ordem os passos da tarefa serão executados

Inferências

- Abstrair
- Atribuir
- Classificar
- Comparar
- Cobrir
- Criticar
- Avaliar
- Gerar
- Agrupar
- Casar
- Modificar
- Operacionalizar
- Propor
- Prever
- Selecionar
- Ordenar
- Especificar
- Verificar

Funções de Transferência



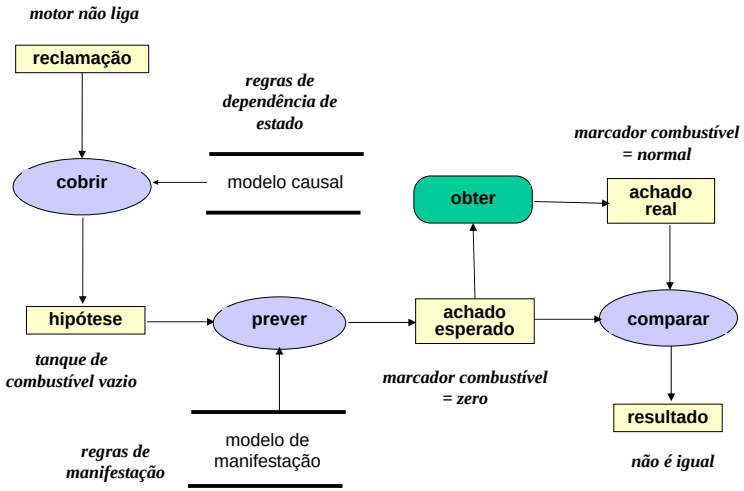
Métodos de Solução de Problemas

- Componente dinâmico do conhecimento
- Modelo abstrato da inferência aplicável àquela classe de problemas
- NÃO correspondem aos métodos de inferência por busca, como raciocínio progressivo ou regressivo
- Generalização de um *padrão* de raciocínio específico, mas *não* é um raciocínio genérico

Tipos de MSPs

- Geração e teste
- Classificação heurística
- Diagnóstico sistemático
- Verificação
- Reparo
- Projeto
- Configuração

Diagnóstico



Modelo da Tarefa Diagnóstico

TASK Diagnóstico;

ROLES:

INPUT:

reclamação:“Queixa do cliente”;

OUTPUT:

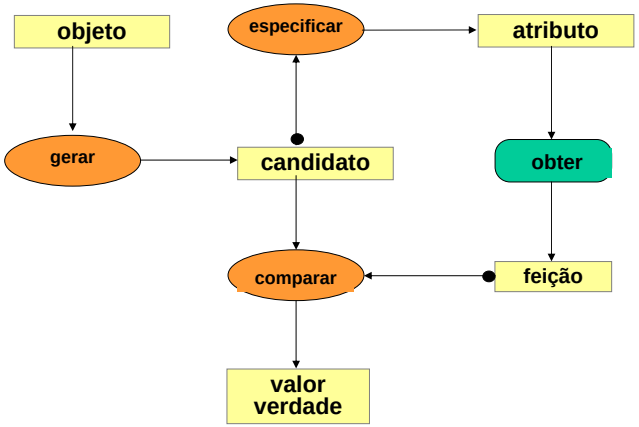
Falhas: “As falhas que causaram reclamações”;

Evidência: “As evidências reunidas durante o diagnóstico”;

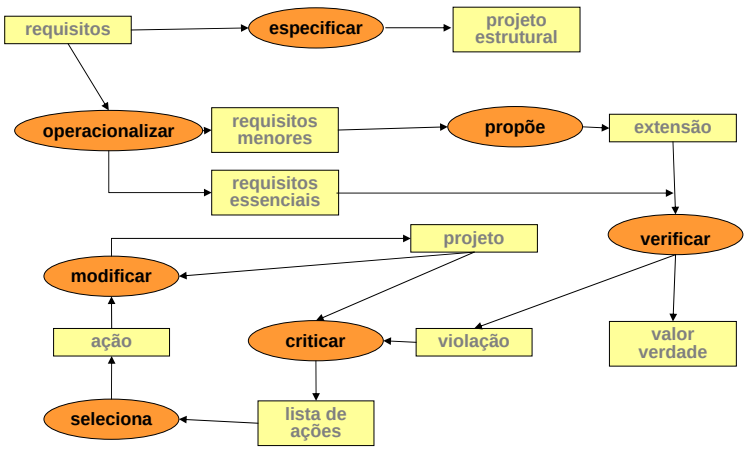
END TASK Diagnóstico;

TASK-METHOD diagnóstico-por-gerar-e-testar;
REALIZES: diagnóstico-de-carro;
DECOMPOSITION:
INFERENCES: cobrir, prever, comparar;
TRANSFER-FUNCTIONS: obter;
ROLES:
INTERMEDIATE:
hipótese: *"uma solução candidata";*
achado-esperada: *"O achado previsto, caso a hipótese seja verdadeira";*
achado-real: *"O achado realmente observado";*
resultado: *"O resultado da comparação";*
CONTROL-STRUCTURE:
WHILE NEW-SOLUTION cobrir(reclamação -> hipótese) DO
prever(hipótese -> achado-esperado);
obter(achado-esperado -> achado-real);
evidência := evidência ADD achado-real;
comparar(achado-esperado + achado-real -> resultado);
IF resultado == equal
THEN "interromper o laço";
END IF
END WHILE
IF result == equal
THEN categoria-falha := hipótese;
ELSE "não foi encontrada uma solução";
END IF
END WHILE

Classificação



Configuração (Propor e Revisar)



Avaliação (ou Assessment)