Kappa-PC: un tool per lo sviluppo di Sistemi Esperti

Ing.Sergio Storari
DEIS Università di Bologna
sstorari@ing.unife.it

Cos'è un Tool

- Tool: strumento che facilità lo sviluppo di un sistema esperto.
- Caratteristiche :
  - Libreria di funzioni che implementano funzionalità più o meno complesse che ogni applicativo e/o sistema esperto necessità: gestione degli oggetti e delle classi, gestione delle stringhe, creazione di regole, ecc
  - Interfaccia grafica di sviluppo
  - Gestione della connessione con i Database
  - Motore di inferenza con caratteristiche peculiari
  - Linguaggio di programmazione general purpose (C++, JAVA) o dedicato
- Non esiste il tool migliore in assoluto ma a seconda della applicazione da realizzare va individuato quello che offre il miglior rapporto Costi/Benefici

K-PC: Cos'è

- Kappa-PC 2.4 della Intellicorp S.p.A.
- Pregi:
  - Ottima interfaccia di sviluppo
  - Programmazione ad oggetti
  - Semplice e ricco di funzionalità
  - Forward e Backward chaining
  - Collegamento ai database
- Difetti:
  - Limitazioni di memoria
  - Applicazione a 16 bit

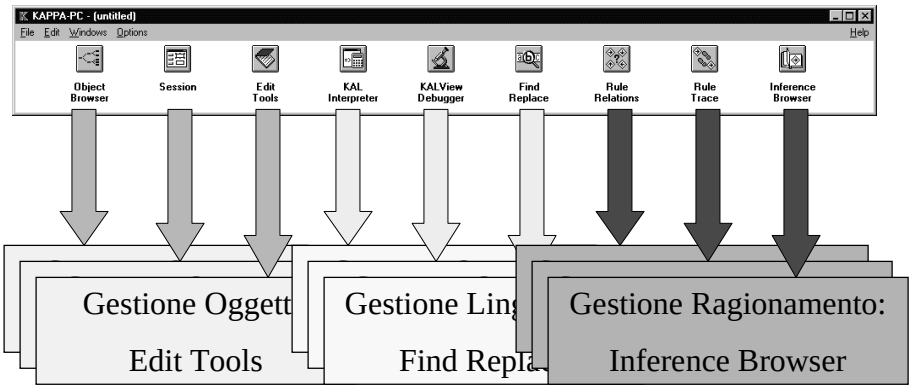
K-PC: Dove trovarlo

- E' installato su tutti i PC del Lab 2
- Il programma contiene un ottimo ed esauriente manuale on-line

Argomenti trattati

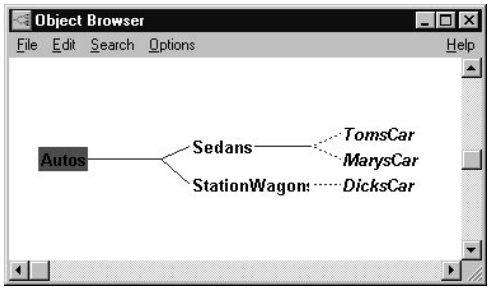
- Base di conoscenza
  - Oggetti
  - Regole
- Motore Inferenziale
  - Goal
  - Forward chaining
  - Backward chaining
- Linguaggio KAL
  - Funzioni

Interfaccia Grafica

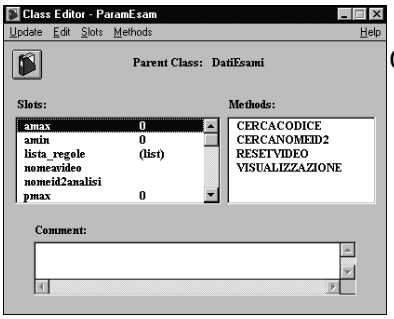


Gli oggetti

- Classi
- Istanze
- Slots
- Ereditarieta` di slot e metodi
  - Se Autos ha uno slot NumberofDoors i discendenti ereditano tale slot



Gestione Oggetti



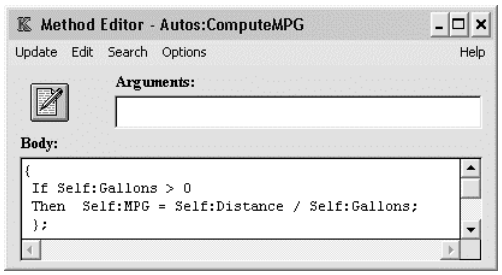
Classi composte da:

- Slot: caratteristiche degli oggetti appartenenti alla classe
- Metodi: funzioni che eseguono azioni sugli oggetti della classe

Gestione Oggetti

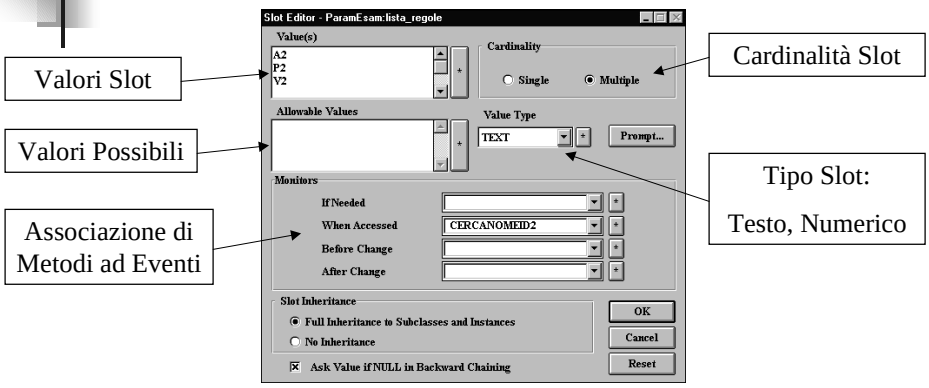
Metodi degli oggetti

- Esprimono in modo procedurale il comportamento degli oggetti
- Sono scritti in linguaggio KAL
- Esistono due argomenti di default: Self e TheParent.
- Esempio:
  - ComputeMPG metodo associato alla classe Autos per calcolare i chilometri per litro dati i chilometri percorsi e la benzina consumata.



Gestione Oggetti

Slot degli oggetti



- Demoni associabili ad eventi
- Scelte sulla ereditarietà

Gestione Oggetti

Monitor

- **If Needed Method:** Contiene il nome di un metodo specificato nell'oggetto od in uno dei suoi predecessori. Se si chiede il valore dello slot e lo slot non ha un valore assegnato si attiva il metodo che riceve come argomento il nome dello slot e ritorna il suo valore.
- **When Accessed Method:** Si esegue il metodo ogni volta che si accede lo slot (anche se a questo e' gia' attribuito un valore).
- **Before Change Method:** Si esegue il metodo prima di inserire un nuovo valore per lo slot. Ha come parametri il nome dello slot ed il suo nuovo valore e ritorna il valore da inserire nello slot.
- **After Change Method:** Si esegue il metodo subito dopo aver inserito un nuovo valore per lo slot. Ha come parametri il nome dello slot ed il suo vecchio valore e puo' eseguire qualunque azione.

Gestione Linguaggio

Linguaggio KAL

- **Atomi:** Una parola, od un insieme di parole fra doppi apici.
  - Esempio: MarysCar o "red white blue"
- **Coppie:** Un nome di un oggetto seguito da ": " ed il nome di uno slot.
  - Esempi: MarysCar:Color Autos:NumberOfDoors
- **Espressioni infisse:** Assegnamento: =, +=
  - Esempi: MarysCar:Speed = 55 TomsCar:Speed += 5;
- **Espressioni aritmetiche**: +,-,*,/,^,<,>,<=,>=,==,!=
- **Operazioni con Testi:** # (appende) #= (uguaglianza fra stringhe)
- **Operatori Logici:** And, Or, Xor, Not **Espressioni speciali**
- **Funzioni di controllo:** While, For, ForAll, If e Let.

Gestione Linguaggio

Linguaggio KAL

- For counter [start end interval] expression;
  Esempio:
  For counter From 1 To 10 By 0.5 Do {
  MakeInstance (Obj#10*counter,Root);
  MakeSlot (Obj#10*counter,Size);
  SetValue(Obj#10*counter,Size,counter);
  };

- ForAll [x | className] expression;
  Esempio:
  ForAll [x | People]
  EnumList (x:Friends,y,
  PostMessage(x # " has a friend "y)));

Gestione Linguaggio

Linguaggio KAL

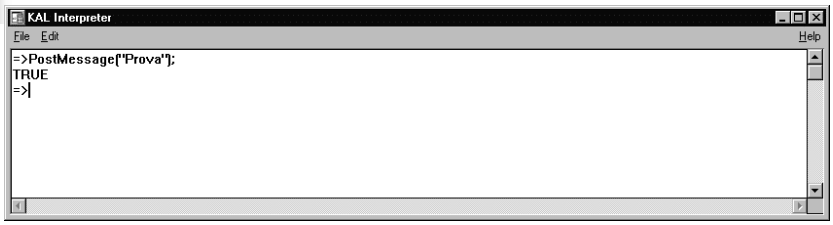
- While (testExpression) expression;
  Esempio:
  while (Car:Fuel > 0)
  SendMessage(Car,Drive,1);

- Let [x xExpression] expression;
  Esempio:
  Let [text From_Car:Origin]
  PostMessage(text);

- if (testExpression)
  Then thenExpression
  Else elseExpression

Gestione Linguaggio

Interprete del linguaggio



- Linguaggio Interpretato
- Il linguaggio KAL può essere convertito in C e successivamente compilato in DLL

Gestione Linguaggio

Interprete del linguaggio

- Ad esempio, per creare una classe:
  ==>MakeClass(Autos,Root);
  Autos

- Per creare una regola:
  ==>MakeRule(Rule1, [car | Autos],
  car:MaxSpeed >=180,
  car:Type=Racing);
  Rule1

- Valutazione di espressioni aritmetiche:
  ==> MyCar:MaxSpeed > 180;
  TRUE

Gestione Linguaggio

Le Funzioni

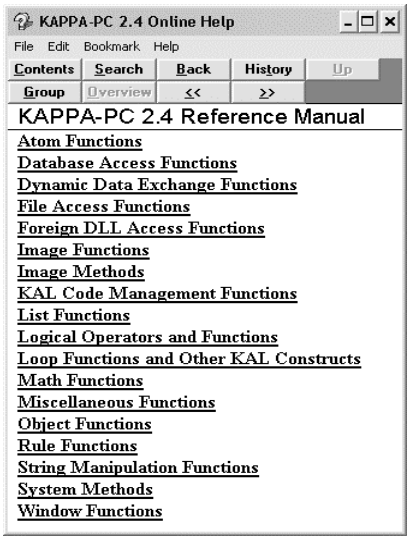
- KAL ha una ricchissima libreria di circa 300 funzioni per:
  - Creare accedere e modificare elementi della KB;
  - Valutare espressioni logiche, matematiche e fra stringhe;
  - Controllare blocchi di espressioni;
  - Manipolare liste, files, databases;
  - Controllare le inferenze;
  - Controllare l'interfaccia grafica.

- Sintassi per la chiamata di una funzione:
  - FunctionName(argument1,....,argumentN);

Gestione Linguaggio

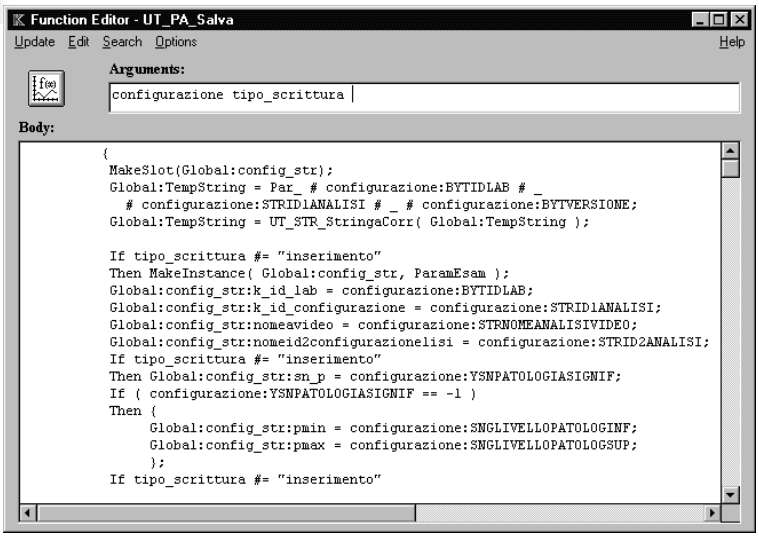
Le Funzioni

- Esempi:
  - **MakeInstance(Used,Autos);** che crea l'istanza Used della classe Autos
  - **PostInputForm("What are the colors", TomsCar, Colors, "Choose One");** che genera una finestra di input in cui l'utente puo' scegliere il colore fra quelli permessi per lo slot Colors dell'istanza TomsCar.



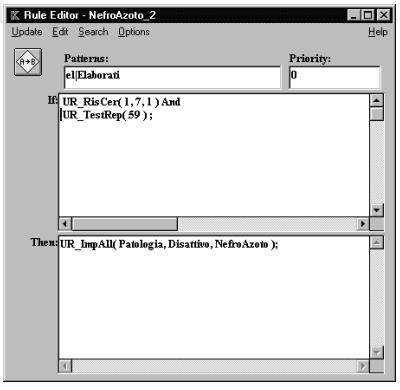
Gestione Linguaggio

Funzioni definite dall'utente



Gestione Ragionamento

Le Regole



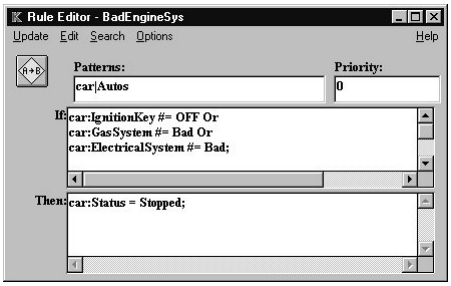
- Pattern Matching
- Priorità
- Definizione di gruppi di regole
- Forward chaining: a partire dai dati
- Backward chaining: a partire dai goal
- Regole di tipo IF … THEN
- Esempio:
- GoodElecSys [car|Autos]

```
If
car:SparkPlugCondition #= Good And
car:Timing #= InSynch And
car:Battery #= Charged;
Then
car:ElectricalSystem = Good;
```

Gestione Ragionamento

Editor regole

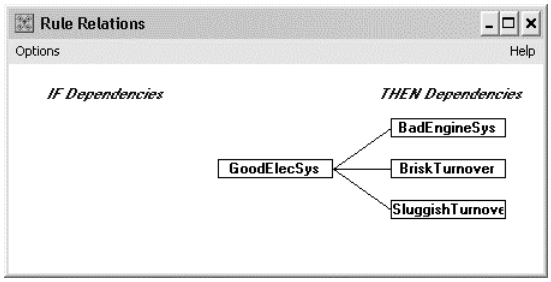
- **Patterns:** Il formato di ogni pattern e' "DummyName | Classname" dove DummyName serve per riferire tutte le istanze di una determinata classe.
- **Priority:** Assegna una priorita' alla regola per risolvere conflitti fra regole (piu' alto e' il numero, maggiore e' la priorita', default 0).
- **If** Espressione KAL
- **Then** Espressione KAL



Gestione Ragionamento

Rule Relation

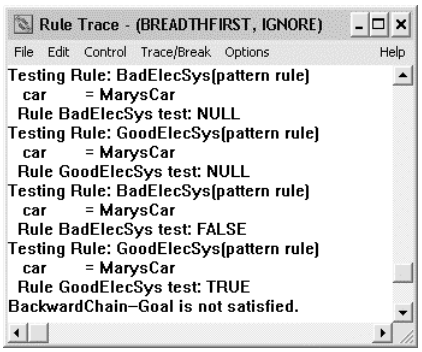
- **Rule Relations Window**
  - Consente di visualizzare le relazioni fra regole.
  - Data una regola R, con If Dependencies mostra le regole che hanno una conclusione che fa match con le premesse di R; mentre con Then Dependencies mostra le regole che hanno premesse che fanno match con le conclusioni di R.



Gestione Ragionamento

Rule Trace

- **Rule Trace Window**
  - Permette di visualizzare il procedimento di ragionamento (sia backward che forward), le nuove conclusioni che il sistema genera e la variazione di alcuni slots.
  - Consente di inserire anche dei breakpoint in certi slots e regole o di fare il trace in modo selettivo.
  - Mostra anche il contenuto dell'agenda e le regole attive.
  - E' possibile anche seguire l'esecuzione step-by-step.



Gestione Ragionamento

Agenda

- Agenda:
  - L'agenda è un insieme di elementi object:slot
  - Gli elementi che cambiano valore a causa della applicazione di una regola sono aggiunti automaticamente alla Agenda
  - Può essere anche usata la funzione Assert
  - Il processo di forward chaining continua finchè ci sono elementi nell'agenda
  - Due modalità:
    - Ignore: un elemento non viene considerato se c'è in agenda una versione di tale elemento più recente
    - NOIgnore: un elemento viene considerato anche se c'è in agenda una versione di tale elemento più recente

Gestione Ragionamento

Motore Inferenziale

- Rilevanza: una regola diventa applicabile se:
  - Un elemento dell'agenda del tipo object:slot verifica una delle premesse AND
  - Tutte le altre premesse dimostrano di essere verificate
- Inferenza:
  - ForwardChain(NULL, Rule1, Rule4, etc...);
  - BackwardChain(Goal1, Global:RuleList1);

Gestione Ragionamento

Forward Chaining

- Il ciclo comincia ogni volta che si estrae una coppia Oggetto:Slot.
- La coppia viene controllata con ogni premessa delle regole presenti nel rule set.
- Si controllano le regole che menzionano tale coppia e si valutano le loro premesse. Se la valutazione dell'espressione risulta vera, la regola viene considerata applicabile (ed inserita nella Rule List).
- Poi si rimuove una regola applicabile dalla Rule List e si esegue valutando le sue conclusioni (che creeranno eventualmente nuove coppie).
- Se le nuove coppie sono soluzione per il Goal, il ciclo forward termina, altrimenti nuove coppie Oggetto:Slot sono inserite nell'agenda in base ai cambiamenti espressi nel conseguente.
- Poi si ricomincia un nuovo ciclo.

Gestione Ragionamento

Risoluzione dei conflitti nel FC

- Strategie impostabili con SetForwardChainMode
- SELECTIVE (default):
  - Segue solo un percorso di ragionamento
  - Le regole che diventano rilavanti vengono aggiunte alla lista delle regole applicabili in accordo con la priorità
- DEPTHFIRST:
  - Strategia esaustiva
  - La lista delle regole applicabili non viene cancellata dopo il successo dell'applicazione di una regola
  - Le regole che diventano rilavanti vengono aggiunte in TESTA alla lista delle regole applicabili in accordo con la priorità. La prima regola di questa lista è quella che viene applicata.
  - Se ci sono elementi nella Agenda e nella lista delle Regole applicabili viene considerata prima la Agenda

Gestione Ragionamento

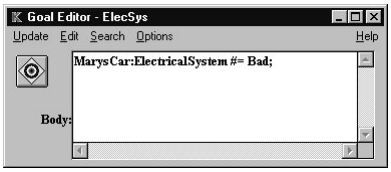
Risoluzione dei conflitti nel FC

- BREADTHFIRST
  - Strategia esaustiva
  - Le nuove regole applicabili sono ordinate a seconda della priorità e aggiunte in coda alla lista delle regole
  - La prima regola che viene applicata può generare nuovi elementi nella agenda
  - Se ci sono elementi sia nella Agenda che nella lista delle regole applicabili si da la precedenza alla lista delle regole
- BESTFIRST
  - Strategia esaustiva simile alla BREADTHFIRST
  - Le nuove regole applicabili sono mescolate a quelle già presenti nella lista delle regole a seconda della priorità.
  - La prima regola che viene applicata può generare nuovi elementi nella agenda
  - Se ci sono elementi sia nella Agenda che nella lista delle regole applicabili si da la precedenza alla lista delle regole

Gestione Ragionamento

Gestione goal

- **Goals**
  - Un goal rappresenta un test
  - E' opzionale nel forward chaining FC e richiesto nel backward chaining BC.
  - Se si verifica nel FC allora si blocca il ragionamento.
- **Goal Editor**
  - Body: Contiene una Espressione KAL che il motore d'inferenza utilizza come Goal da dimostrare.



Gestione Ragionamento

Backward Chaining

- Si parte da un Goal che deve essere verificato e si possono avere tre fasi:
- Espansione:
  - In questa fase il motore di inferenza cerca di valutare o la parte If delle regole o la coppia oggetto:slot al fine di soddisfare il goal. L'espansione agisce da sinistra verso destra
- Collassamento:
  - Il collassamento del goal è quella fase in cui il motore di inferenza tenta di determinare se il goal è soddisfatto.
  - Il goal potrebbe essere soddisfatto da nuovi fatti aggiunti all'agenda durante la fase di Espansione.
- Richiesta:
  - Il motore di inferenza chiede all'utente i valori degli slot per i quali non riesce a valutare il valore da sola.

Gestione Ragionamento

Esempio

- Regole usate:
  - BriskTurnover:
    - [car:Autos]
      IF car:IgnitionKey #= ON And
      car: ElectricalSystem #= Good;
      THEN car:EngineTurnover = Brisk;
  - SluggishTurnover:
    - [car:Autos]
      IF car:IgnitionKey #= ON And
      car: ElectricalSystem #= Bad;
      THEN car:EngineTurnover = Sluggish;
  - NoTurnover:
    - [car:Autos]
      IF car:IgnitionKey #= OFF;
      THEN car:EngineTurnover = Absent;

Gestione Ragionamento

Esempio

- **Slots utilizzati:**
  - MarysCar: IgnitionKey
    - Allowable Values: ON OFF
  - MarysCar: ElectricalSystem
    - Allowable Values: Good Bad
  - MarysCar: EngineTurnover
    - Allowable Values: Brisk Sluggish Absent
- **Definizione del goal:**
  - Name: Turnover?
  - Body: MarysCar: EngineTurnover #= Brisk;
- **Definizione del Rule Set**
  - MakeSlot(Global,EngRules,BriskTurnover,NoTurnover, SluggishTurnover);

Gestione Ragionamento

Esempio

- **Visualizzazione del processo di Backward Chaining**
  - Si puo' attivare e visualizzare il procedimento mediante l'Inference Browser, scegliendo l'opzione Step mode (oppure si puo` attivare dal Rule Trace Window).
  - Nel campo Arguments si inserisce: Turnover?, Global:EngRules
- Prima prova con la regola (BreadthFirst search):
  - BriskTurnover:
  - Deve chiedere i valori delle premesse perche' non sono conseguenze di alcuna regola che puo' diventare vera e non sono ancora determinati nella base di conoscenza.
    - MarysCar: IgnitionKey (risposta ON)
    - MarysCar: ElectricalSystem (risposta GOOD)
    - Risposta: TRUE
- Si deve inserire in Arguments: [NOASK],Turnover?, Global:EngRules
- Risultato NULL perche` si e` inibita la richiesta di valori all'utente.