Sistemas Baseados em Regras
Aula3: Drools

Profa. Patrícia Dockhorn Costa

pdcosta@inf.ufes.br

www.inf.ufes.br/~pdcosta/ensino

Drools

- “Business Logic integration Platform”
- Plataforma integrada para gerenciamento de:
  - Regras
  - Workflow
  - Eventos

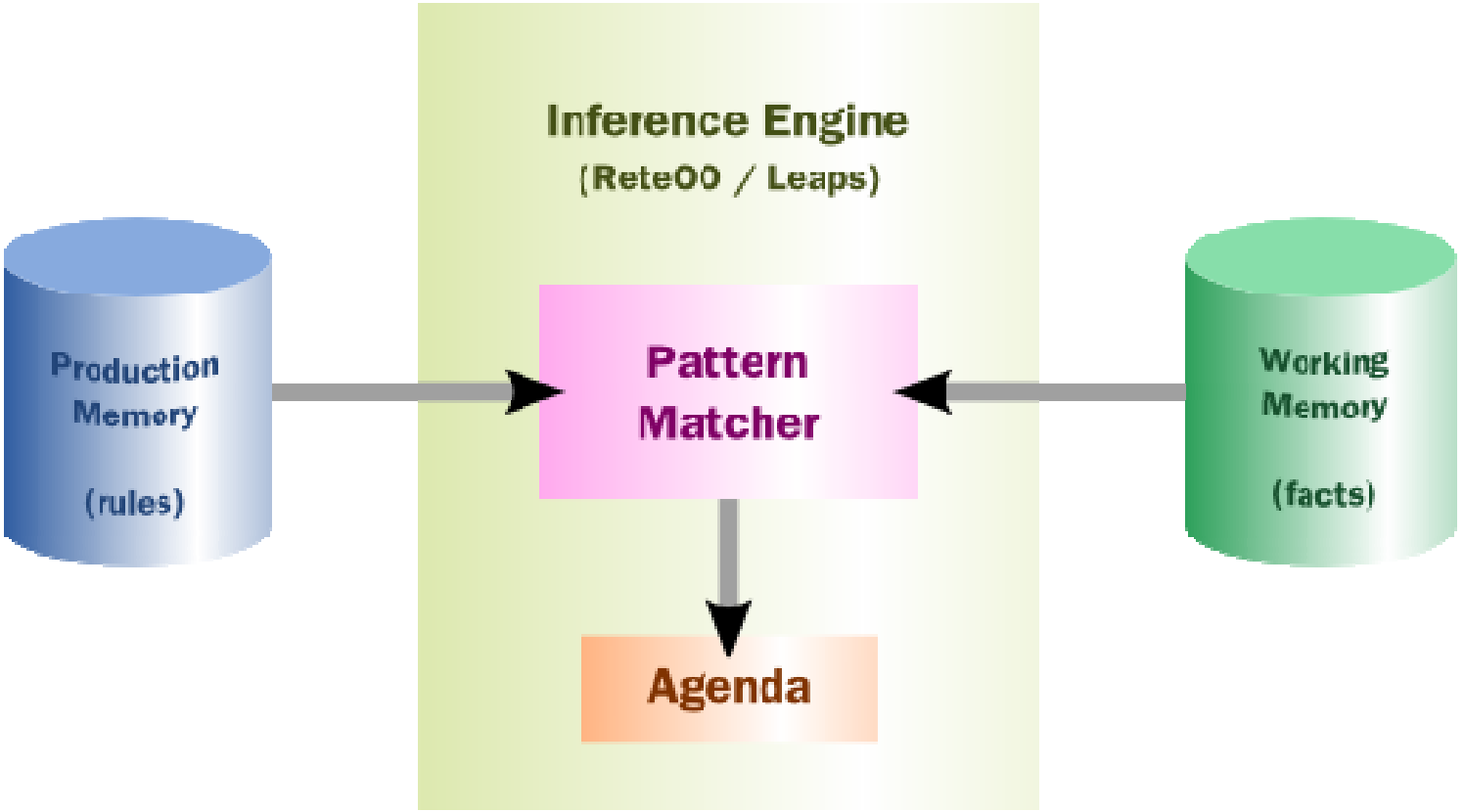
Plataforma Drools

- Drools Guvnor (business rules manager)
- Drools Expert (rule engine)
- jBPM 5 (process/workflow)
- Drools Fusion (event processing/temporal reasoning)
- Drools Planner (automated planning)

Drools Expert

Rule Type	Production
Execution Method	Forward Chaining
Pattern Matching	Rete Algorithm
Conflict Resolution	Salience and LIFO

Arquitetura do Drools Expert



Fatos em Drools

- Fatos são objetos java que podem ser acessados por uma regra
- A máquina de regras mantém apenas referencias para os objetos
- Para que objetos tenham referência na máquina, devemos definir *getters* e *setters* nas respectivas classes (ou definir diretamente na working memory usando *declare*)

Um exemplo de tipo de fato (em Java)

```
public class Applicant {
      private String name;
      private int age;
      private boolean valid;
      //getter and setter methods
   here
}
```

Ou diretamente no arquivo de regras ...

```
Declare Applicant
     name: String
     age: int
     valid: boolean
end
```

Um exemplo de regra ...

```
rule "Is of valid age"
when
  $a : Applicant( age < 18 )
then
  $a.setValid( false );

end
```

Um pouco mais sobre a regra...

- Define duas “constraints”:
  - Tipo (Applicant)
  - Valor de campo (age < 18)
- Constraint de tipo (com ou sem constraints de campo) é chamado de Padrão (Pattern)
- Quando uma instância inserida satisfaz as constraints (de tipos e campos), há um Casamento (data is Matched)

Working Memory

- Armazena os fatos
- Fatos podem ser inseridos, modificados ou removidos
- Permite:
  - Executar *queries*
  - Acesso a *Entry Points*

Working Memory Entry Point

- Insertion (assertion)
  - Insere um fato na working memory
  - Momento da avaliação das regras (pode ativar regras)
- Retraction
  - Remove um fato da working memory
  - Pode cancelar ativação de regras
  - Pode ativar regras (regras que dependem da não existencia de um fato)

Working Memory Entry Point (cont.)

- Update
  - Notifica a working memory de alguma mudança no objeto
  - Internamente é tratado como um retraction seguido de um insertion

Insertion Modes

- Determina como os fatos são gerenciados pela working memory:
- Identity insertion
  - Novos fatos são comparados a fatos existentes por “referência”
  - Novo FactHandle é retornado se não houver referência do objeto na working memory
- Equality insertion
  - Novos fatos são comparados a fatos existentes por “valor”
  - Se o conteúdo do objeto for diferente, retorna um novo FactHandle

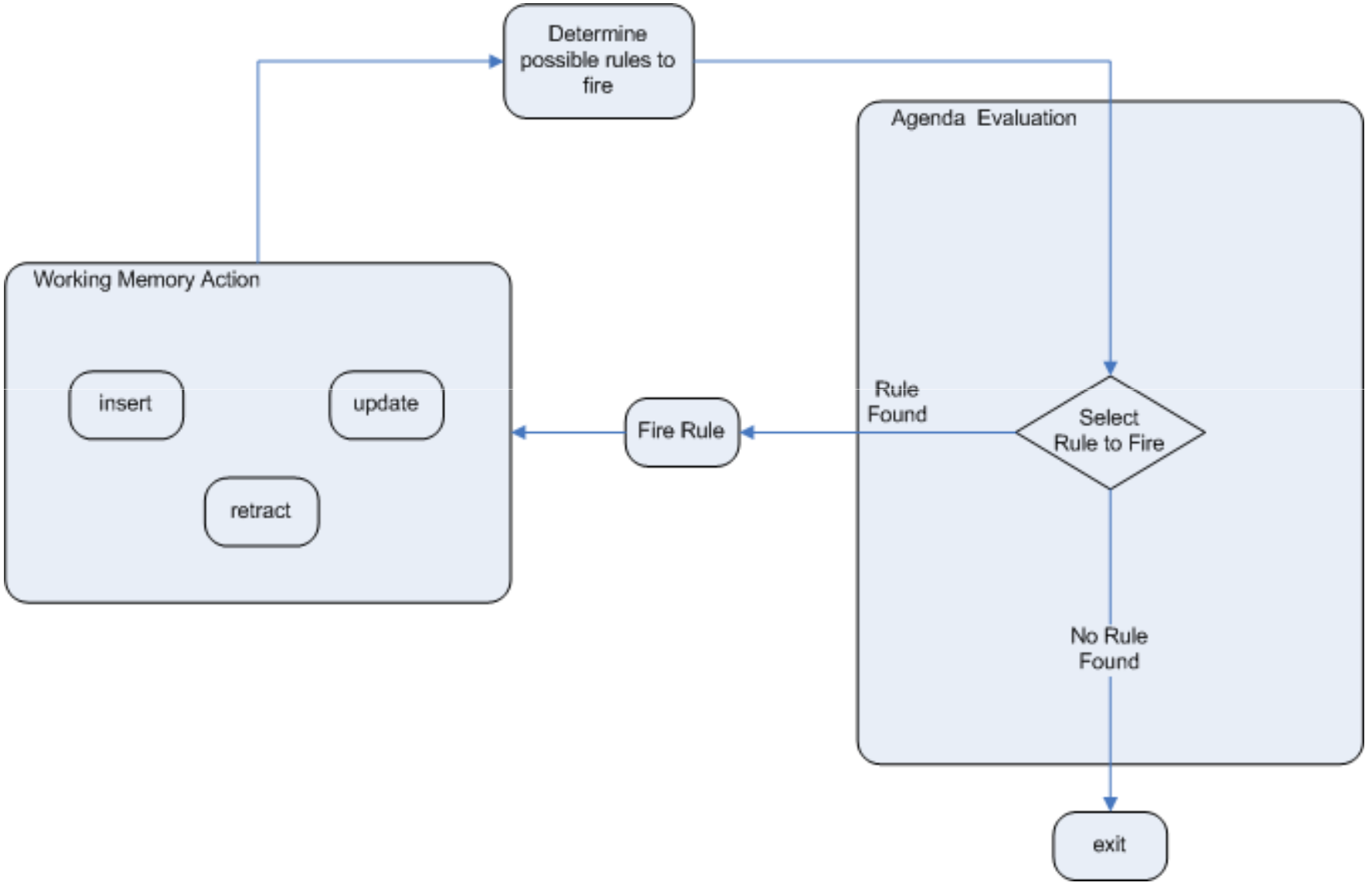
Production Memory

- A *Production Memory* armazena as regras
- O processamento das regras é definido pelo algoritmo *Rete*, no processo de Pattern Matching (casamento de padrões).
- Drools usa o ReteOO, que é uma extensão do Rete

Agenda

- Controla a ordem de execução das regras ativadas (já casadas) usando a estratégia de resolução de conflitos (Salience or LIFO)

Engine Cycles



Truth Mantainance

- Responsável por manter e restaurar a consistência da working memory
- Fatos inseridos “logicamente” são automaticamente retirados quando as condições que o inseriram não forem mais verdadeiras
- Uma insertion de fato pode ser:
  - *Stated*: inserção normal.
  - *Justified:* inserção lógica (somente na RHS de uma regra).

Stated Insertions

- Baseado no modo de inserção “identity”. Ou seja, podem existir instâncias iguais
- Quando uma nova instância é inserida, the “stating counter” é incrementando

Justified Insertions

- Baseado no modo de inserção “equality”. Ou seja, apenas uma instância por objeto
- Inserções subsequentes incrementam o “justification counter”
- Quando uma “justification” é removida (o LHS fica falso), o contator é decrementado.

A Linguagem de Regras

- Regras são escritas em lógica de primeira ordem (ou lógica de predicados)
- *Proposições* são expressões avaliadas como falsas ou verdadeiras
  - Podem ser “open statements” ou “closed statements”
- Em Java, proposições são do tipo “variable operator value”
  - *Value* geralmente sendo literal
- Neste contexto, proposições são “field constraints”

A Linguagem de Regras (cont.)

- Proposições podem ser co
  conectivos conjuntivos e

Duas proposições do tipo "open statements" conectadas por um conectivo disjuntivo

```
person.getEyeColor().equals("blue") ||
  person.getEyeColor().equals("green")
```

A Linguagem de Regras (cont.)

- Em regras...

```
Person( eyeColour == "blue" ) || Person(
  eyeColor == "green" )
```

Usando um “Conditional Element” conectivo disjuntivo – na verdade resulta na generação de duas regras, que representam os 2 possíveis resultados lógicos

A Linguagem de Regras (cont.)

- Em regras...

```
Person( eyeColour == "blue"||"green" )
```

Usando o conectivo disjuntivo para restrição de campos .
Não resulta na geração de múltiplas regras

Expressividade...

- Lógica de primeira ordem é “Turing complete”
- Permite a utilização de quantificadores universais e existencias
- Quantificador Universal
  - Permite verificar a veracidade de algo para um conjunto
  - Geralmente “forall” conditional element
- Quantificador Existencial
  - Permite verificar existencia de algo
  - “not” e “exists” conditional elements

Exemplo em Java...

- Java é “Turing complete”
- Pode-se escrever código para iterar em estruturas de dados para verificar existencia

```
List failedStudents = new ArrayList();
for ( Iterator studentIter = students.iterator();
   studentIter.hasNext() {
   Student student = ( Student ) studentIter.next();
     for (Iterator it=student.getModules.iterator();
       it.hasNext(); ) {
       Module module = ( Module ) it.next();
       if ( module.getScore() < 40 ) {
             failedStudents.add( student ) ; break;
} } }
```

Exemplo em SQL...

```
select *
from Students s
where exists (
  select *
  from Modules m
  where
  m.student_name = s.name and
   m.score < 40
```

Em Regras...

```
rule "Failed_Students"
when
exists( $student : Student() &&
  Module( student == $student,
  score < 40 ) )
```