Apache UIMA™ Ruta
Rule-based Text Annotation
Version 2.1.0

http://uima.apache.org/ruta.html

Peter Klügl
**What is UIMA Ruta?**

Rule-based script language interpreted by a generic Analysis Engine

- **Ruta Engine**
- **Analysis Engine**
- **Dictionary**
- **Type System**
- **Script**

Eclipse-based development environment and tooling:
**UIMA Ruta Workbench**

---

2008: First TextMarker release on SourceForge  
2011: TextMarker contributed to Apache UIMA  
2013: Renamed to UIMA Ruta  
2013: Version 2.1.0 released
Agenda

I. UIMA Ruta Language
II. UIMA Ruta Workbench
Annotation seeding

- Provide some initial annotations
- Seeding is extensible
(Simplified) Script Syntax

**Script** → Package? Import* Statement*

**Import** → ("TYPESYSTEM" | "SCRIPT" | "ENGINE" | "UIMAFIT") Identifier ";"

**Statement** → (Declaration | Rule | Block)

**Block** → "BLOCK" "(" Identifier ")" RuleElement "{" Statement+ "}"
(Simplified) Rule Syntax

Rule → (RuleElement+ | RegExpRule | ConjunctRules) “;”


MatchReference → (TypeExpression | StringExpression | ComposedRE | WildCard)

ComposedRE → “(” RuleElement ((“&” | “|”)? RuleElement)* “)”

TypeExpression (MatchReference)
  ↓ Condition
  ↓ Action
  ↓ Action
  ↓ Action
  ↓ Condition
  ↓ Action

ANY{INLIST(MonthsList) -> MARK(Month), MARK(Date,1,3)} PERIOD? NUM{REGEXP(".\{2,4}\") -> MARK(Year)};

RuleElement

TypeExpression (MatchReference)
  ↓ Quantifier
  ↓ Action

(Animal (COMMA | SEMICOLON)*)+{-> MARK(AnimalEnum,1,2)} Animal;

ComposedRE

RuleElement
Rule Syntax: Syntactic Sugar

• **MatchReference:** FeatureExpression, FeatureMatchExpression

```java
Dependency.governor CW; Token.pos.begin == 0;
```

• **Implicit Actions:** TypeExpression, FeatureAssignmentExpression

```java
Paragraph{CONTAINS(B) -> Headline}; Paragraph{ -> Paragraph.begin == 0};
```

• **Implicit Conditions:** BooleanExpression, FeatureMatchExpression

```java
CW{CW.begin > 10}; CW{boolVar -> MARK(SomeType)};
```

---

`ANY{INLIST(MonthsList) -> MARK(Month), MARK(Date,1,3)} PERIOD? NUM{REGEXP(".\{2,4\}" -> MARK(Year));
```

*TypeExpression instead of MARK action*

```java
(ANY{INLIST(MonthsList) -> Month} PERIOD? NUM{REGEXP(".\{2,4\}" -> Year}){ -> Date};
```
Rule Inference

Basic algorithm:
1. Find valid positions for first rule element
2. Evaluate if following rule element can match next to previous position (repeat for all rule elements)
3. Apply actions if complete rule successfully matched

- Composed rule elements delegate to their inner elements
- Quantifiers specify how often rule element matches:

\[
? \quad ?? \quad * \quad *? \quad + \quad +? \quad [1,2] \quad [1,2]?
\]

\[
\text{ANY\{INLIST(MonthsList) \rightarrow MARK(Month), MARK(Date,1,3)\} PERIOD? NUM\{REGEXP(".\{2,4\}\)\) \rightarrow MARK(Year)}\};
\]

Rule Inference

- Imperative rule execution
- Based on complete disjoint partitioning (RutaBasic)
- Depth-first matching
  - Complete current alternative before matching the next one
    
    \[
    \text{ANY+\{-PARTOF(Text) \rightarrow Text\};} \quad \text{PERIOD Annotation PERIOD;}
    \]
- Only permutations in matching direction
- Manual selection of starting rule element

\[
\text{ANY LastToken;} \quad \text{ANY @LastToken;}
\]
- Dynamic anchoring: Guess best rule element
- Special rule element: „do not care“ wildcard

\[
\text{PERIOD ANY+?\{-\rightarrow Sentence\} PERIOD;} \quad \text{PERIOD \#\{-\rightarrow Sentence\} PERIOD;}
\]
Beyond Sequential Patterns

• Conjunctive rule elements
  – All rule elements need to match at same position
  – Use largest match to continue

```plaintext
(Token.posTag=="DET" & Lemma.value=="the");
NUM (W{REGEXP("Peter") -> Name} & (ANY CW{PARTOF(Name)}));
```

• Disjunctive rule elements
  – One rule element needs to match

```plaintext
(Animal ((COMMA | "and") Animal)+){-> AnimalEnum};
(("Peter" CW) | ("Mr" PERIOD CW)){-> Name};
```

• Conjunct rules
  – Both rules need to match anywhere in window

```plaintext
CW NUM % SW NUM{-> MARK(Found, 1, 2)};
```
Imports, Declarations and Expressions

- Supported imports
  - Scripts
    ```java
    SCRIPT uima.ruta.example.Author;
    ```
  - Type Systems
    ```java
    TYPESYSTEM utils.PlainTextTypeSystem;
    ```
  - Analysis Engines
    ```java
    ENGINE utils.PlainTextAnnotator;
    ```
  - uimaFIT Analysis Engines
    ```java
    UIMAFIT de.tudarmstadt.ukp.dkpro.core.tokit.BreakIteratorSegmenter;
    ```

- Supported declarations:
  - Types
    ```java
    DECLARE SimpleType1, SimpleType2;
    DECLARE ParentType NewType (SomeType feature1, INT feature2);
    ```
  - Variables for types, int, strings, booleans, lists, ...
  - Resources
    ```java
    WORDLIST listName = 'someWordList.txt';
    WORDTABLE tableName = 'someTable.csv';
    ```

- Supported Expressions
  - Primitive types, variables, functions
  - String concatenations, boolean comparison, operations on numbers, ...

- All arguments of conditions/actions are expressions
Actions, Conditions and Functions

Language provides right now:

• 41 Actions
  – MARK, UNMARK, CREATE, TRIE, TRIM, SHIFT, CALL, EXEC, ...

• 27 Conditions
  – CONTAINS, PARTOF, REGEXP, STARTSWITH, NEAR, ...

• Functions for Type, Boolean, String and Number

• Extensible language definition: Add your own elements
Filtering and Visibility

- Complete document is modelled
- No restriction to tokens
- Invisible types = (default + filtered) – retained
- Annotations are invisible, if their begin or end is invisible
- Filtering adaptable by rules
- Make uninteresting parts invisible (default: space, markup, …)

```
Sentence;
Document{-> RETAINTYPE(SPACE)};
Sentence;
Document{-> FILTERTYPE(CW)};
Sentence;
Document{-> RETAINTYPE, FILTERTYPE};
```

Matches on:
- Dec<br>2004, July 85, May1999
- May1999
- May1999
Blocks and Inlined Rules

• BLOCK construct
  – Modularize scripts beyond files
  – Conditioned statements
  – Windowing
  – Foreach loops
  – Procedures (recursion)

• Inlined Rules
  – As „actions“:  ->
    • Simplified block constructs
  – As „conditions“:  <-
    • Nested conditions
Scoring Rules

- ... for dealing a bit with uncertainty
- ... for weighting different aspects
- Action **MARKSCORE** adds score
- Condition **SCORE** validates score against a threshold

```plaintext
STRING s;
Paragraph{CONTAINS(W,1,5) -> MARKSCORE(5,HeadlineInd)};
Paragraph{CONTAINS(W,6,10) -> MARKSCORE(2,HeadlineInd)};
Paragraph{CONTAINS(Bold,80,100,true) -> MARKSCORE(7,HeadlineInd)};
Paragraph{CONTAINS(Bold,30,80,true) -> MARKSCORE(3,HeadlineInd)};
Paragraph{CONTAINS(CW,50,100,true) -> MARKSCORE(7,HeadlineInd)};
Paragraph{CONTAINS(W,0,0) -> MARKSCORE(-50,HeadlineInd)};
HeadlineInd{SCORE(10) -> MARK(Headline)};
HeadlineInd{SCORE(5,10) -> MATCHEDTEXT(s), LOG("Maybe a headline: " + s)};
```
Simple RegExp Rules

- Match on regular expressions (supports capturing groups)
- No restrictions due to partitioning or visibility

DECLARE T1, T2;

DECLARE Annotation Complex (STRING s, Annotation a, BOOLEAN b);

"A(.*?)C" -> T1, 1 = T2;

"B(.*?B(\.)" -> 1 = Complex ("s" = 0, "a" = 2, "b" = true),
              2 = Complex ("s" = 0, "a" = 1, "b" = false);

RegExpRule → StringExpression "->" GroupAssignment
              ("," GroupAssignment)* ";"
GroupAssignment → TypeExpression FeatureAssignment?
               | NumberExpression "=" TypeExpression FeatureAssignment?
FeatureAssignment → "(" StringExpression "=" Expression
                   ("," StringExpression "=" Expression)* ")"
Analysis Engines and Type systems

• **UIMA Ruta**
  – BasicEngine.xml (RutaEngine.class) (includes TypePriorities.xml)
  – BasicTypeSystem.xml (includes InternalTypeSystem.xml)

• **Additional Analysis Engines shipped with UIMA Ruta**
  – AnnotationWriter
  – Cutter
  – HtmlAnnotator (with Type System)
  – HtmlConverter
  – Modifier
  – PlainTextAnnotator (with Type System)
  – ViewWriter
  – XMIWriter
Configuration Parameters

Control of Analysis Engine
- mainScript
- scriptPaths
- descriptorPaths
- additionalScripts
- ...

Explanation of Analysis Engine
- debug
- profile
- ...

This section lists all configuration parameters, either as plain parameters, or as part of one or more groups. Select one to show, or set the value in the right-hand panel.
Agenda

I. UIMA Ruta Language

II. UIMA Ruta Workbench
UIMA Ruta Workbench: IDE for UIMA Ruta rules

- Full-featured rule editor
  - Syntax highlighting
  - Semantic highlighting
  - Syntax checking
  - Auto-completion
  - Template-based completion
  - Open declaration
  - Formatter
- Generates descriptors for scripts
  - Analysis Engine
  - Type System
- Many useful tools
- Supports Mixin-Workspaces
  - Dependencies to Java projects
UIMA Ruta
Project Layout

Script files

Own descriptors

Generated descriptors

Additional components

Template descriptors

Default folders for launch config

Word lists
UIMA Ruta Explain Perspective

What rule created this annotation?

Complete listing of rule applies:
- How often tried
- How often succeeded
- Profiling information

Where did this rule match or fail?

Why did this rule (not) match on this position?
Ruta Query View

• Use rules as query statements
• View displays rule matches in a set of documents
Annotation Testing View

- Unit tests for UIMA Ruta scripts
- Compare result of rules against gold documents
- True Positives, False Positives and False Negatives are displayed in CAS Editor
- Different engineering approaches
  - Create gold document → test-driven development of rules
  - Store correct result of rules as test → backtesting when rules are extended/modified

Gold documents

Evaluation results
Constraint-driven Evaluation (CDE)

- Formalize expectations on domain as constraints:
  - with UIMA Ruta rules
  - with Annotation Distributions
- Test rules (the output) on expectations
- No labeled data needed
- ... but can be used to develop constraints
- Greatly improves rule engineering
Check Annotations view

- Support creation of gold documents
- Control for CAS Editor
  - Specify type system
  - Specify highlighted types
  - Select annotation mode
- Remembers processed types
TextRuler: Framework for rule learning algorithms

Available algorithms

Target types

Gold data and features

Result
TextRuler: Available algorithms

- **LP² [Ciravegna, 2003]**
  - Boundary matching rules
  - Context rules for filling gaps in boundary rules
  - No correction rules (yet)

- **Whisk [Soderland et al., 1999]**
  - Rules in the form of modified regular expressions
  - No multi-slot rules (yet)

- **KEP**
  - Basic idea: How does a human write rules?
  - Set of simple algorithms for different engineering patterns
  - Exploit synergy by combination

- **Trabal**
  - Transformation-based rule learner
  - Try to learn rules that correct existing annotations
Preferences and Popup Commands

Preferences

Apply script on folder

Compile word lists

Update UIMA Ruta Project
Summary

Apache UIMA Ruta
Rule-based Text Annotation

http://uima.apache.org/ruta.html

- Compact and powerful language for text processing tasks
- Supports different approaches
- Designed for rapid development
- Extensible and combinable
- Serious engineering support