



ERESYE: Artificial Intelligence in Erlang Programs

*Antonella Di Stefano, +Francesca Gangemi, *Corrado Santoro

*University of Catania, Italy

+Erlang Training and Consulting, London, UK

ACM Erlang Workshop – Tallinn, Sept. 25, 2005

Overview



- Motivations
- Background
 - Expert Systems/Rule production systems
 - CLIPS and the RETE algorithm
- ERESYE
 - Rule model
 - ERESYE Internals
 - Knowledge and ontology representation
- CLIPS/ERESYE Comparison
- Conclusions

Motivations



- Traditional languages (Java, C, C++) cannot be directly used when “intelligent capabilities” have to be included in a system
- In these cases, an external tool has to be integrated, which is programmed using a different language (LISP, Prolog, etc.)
 - Lack of implementation homogeneity
 - Two different programming philosophies and approaches
- Erlang looks promising since
 - it is general-purpose (like Java, C, C++)
 - **atoms** and **tuples** are well-suited to represent a *knowledge*
 - function clauses fit well in the representation of production rules
 - pattern matching facilitates the implementation of rule-handling algorithms

Background



- An expert system provides solutions based on *knowledge* and *experience*
- Expert systems are often programmed by means of *production rules* manipulating *facts*
- CLIPS (and its Java-based clones, JESS and Drools) is one of the most widely used tool
- All of these systems are based on RETE, a fast and effective “many-patterns/many-objects” matching algorithm
- They use a LISP-like syntax where a fact is represented as a list of terms enclosed in parentheses:

(color pen blue)

CLIPS Examples



```
(defrule convert
  ?temp <- (temperature ?x F)
=>
  (retract ?temp )
  (assert (temperature (+ ( - ? x 32) (/ 5 9) C ))))
```

Rule Definition

Precondition

Action

```
(defrule move (declare salience 99)
  (on_table ?object )
  ((color ?object red ) or (color ?object blue ))
  => (assert (move_object ?object ))
      (retract (on_table ?object ) )
```

Rule priority

- Rules are triggered by the presence, in the KB, of one or more facts matching a given predicate (**precondition**)
- The action can be any computation, including retracting an existing fact or asserting a new fact (**action**)

Executing Rules: the RETE Algorithm



- It is used to match the precondition of a rule
- The precondition is transformed in rooted directed acyclic graph (a *network*)
- Network nodes perform test on fact patterns:
 - 1-input nodes (**alpha-nodes**) perform tests on individual facts
 - 2-input nodes (**join-nodes**) perform tests for consistent variable binding between two or more fact patterns
- Each node holds a *memory*
 - **Alpha-memories** store facts satisfying the alpha-node's condition
 - **Beta-memories** store sequence of facts matching some pattern facts in a rule
- When a fact is asserted (or retracted), a **token** is passed to the network's root
 - For each node, the node's condition is tested
 - If test succeeds, the token is propagated
 - If the token reaches the bottom of the network, then the rule is fired

An example

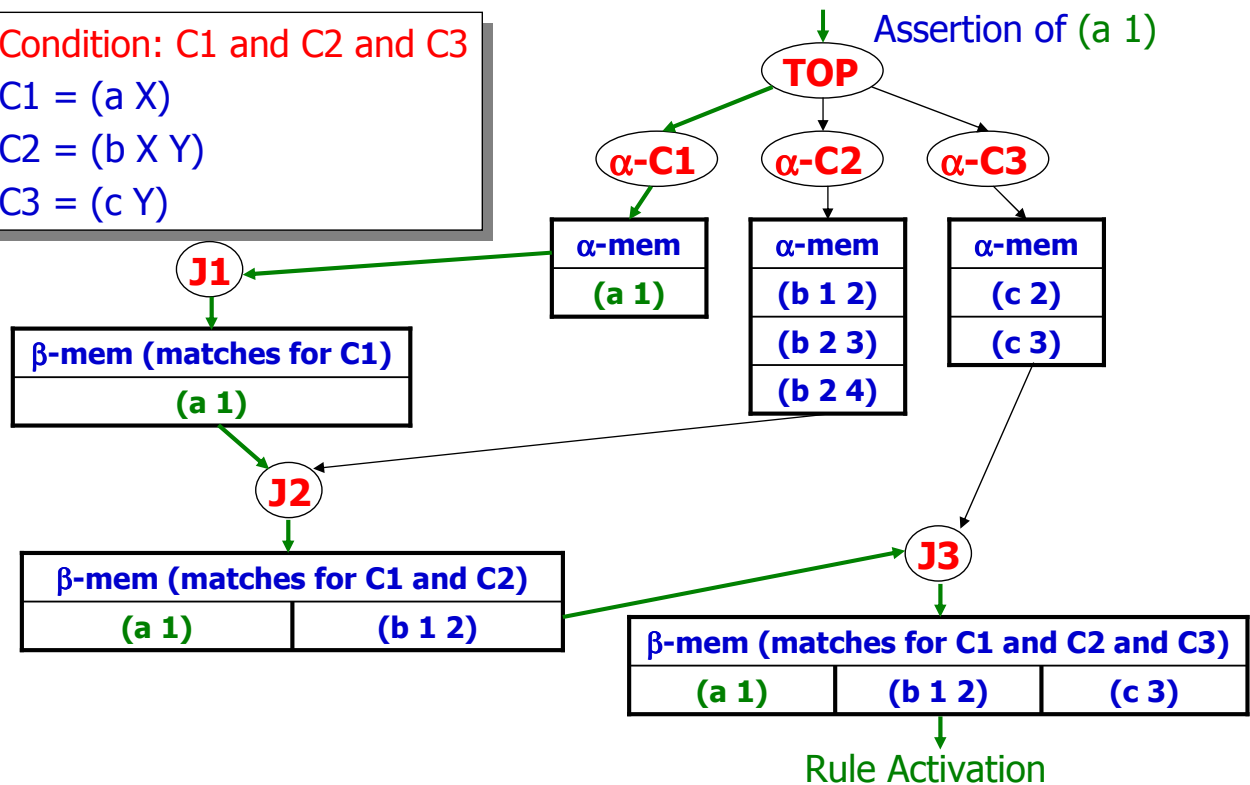


Condition: C1 and C2 and C3

C1 = (a X)

C2 = (b X Y)

C3 = (c Y)

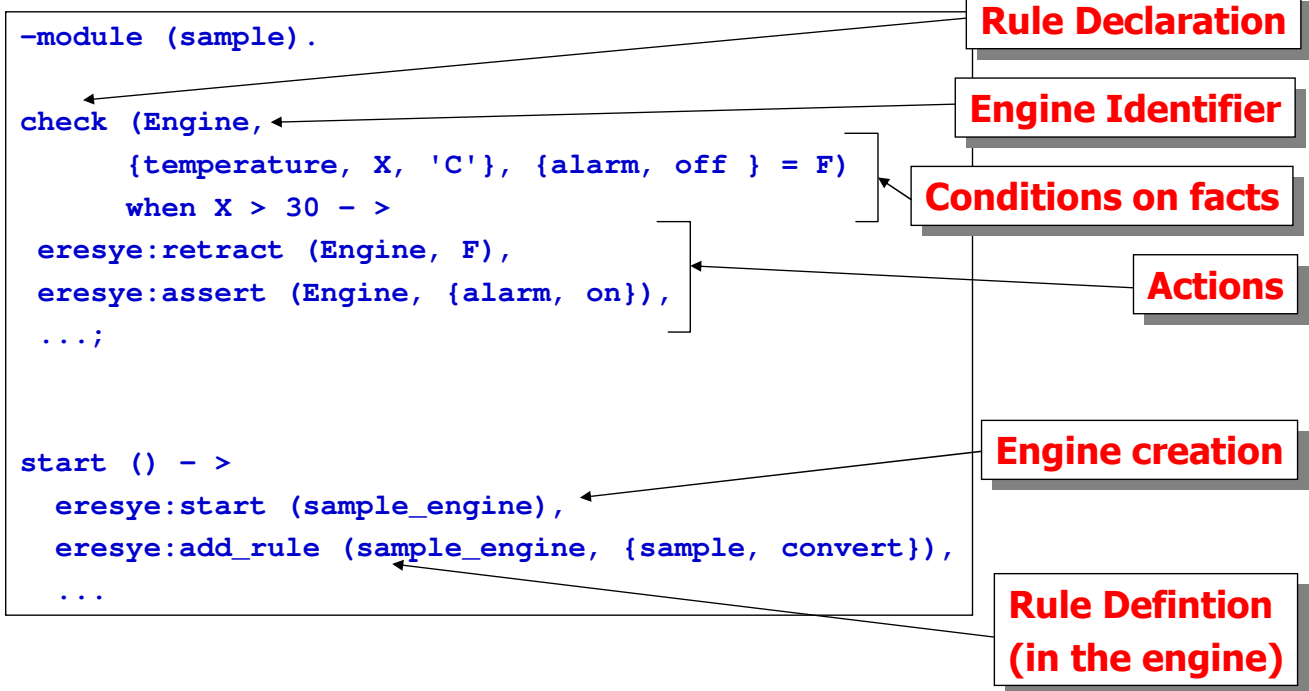


ERESYE

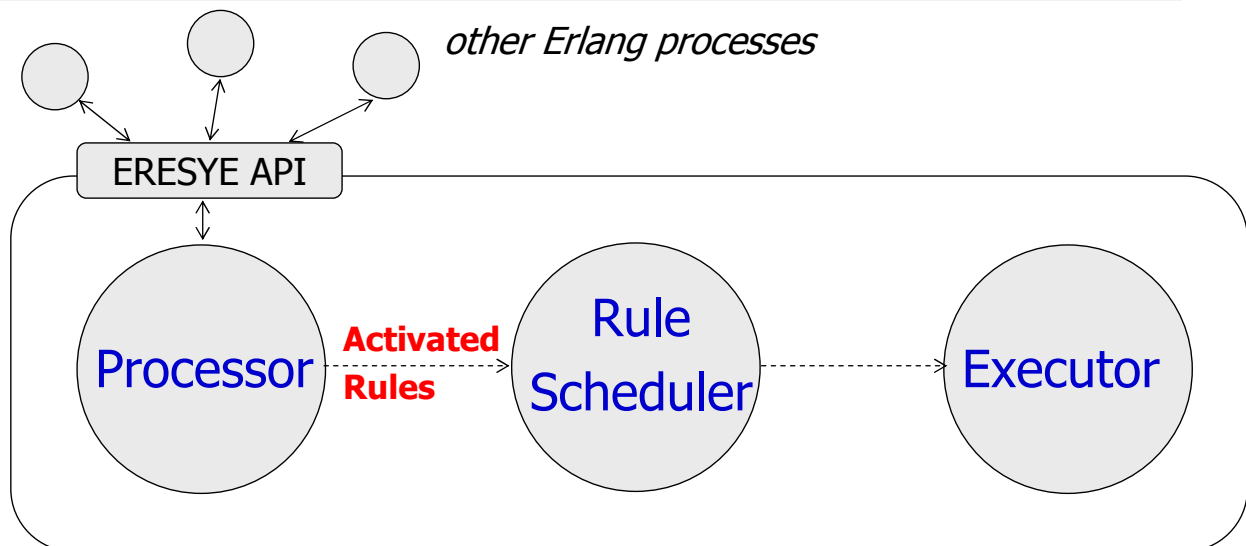


- It allows the creation and execution of multiple rule processing engines in Erlang
- Each engine has a knowledge based and a set of rules
- Facts (knowledge) are represented using Erlang tuples, e.g.
 - {temperature, 50, 'F'}
 - {alarm, on}
 - {speed, 30, 'km/h'}
- Inference rules are written using standard Erlang functions

An ERESYE Program

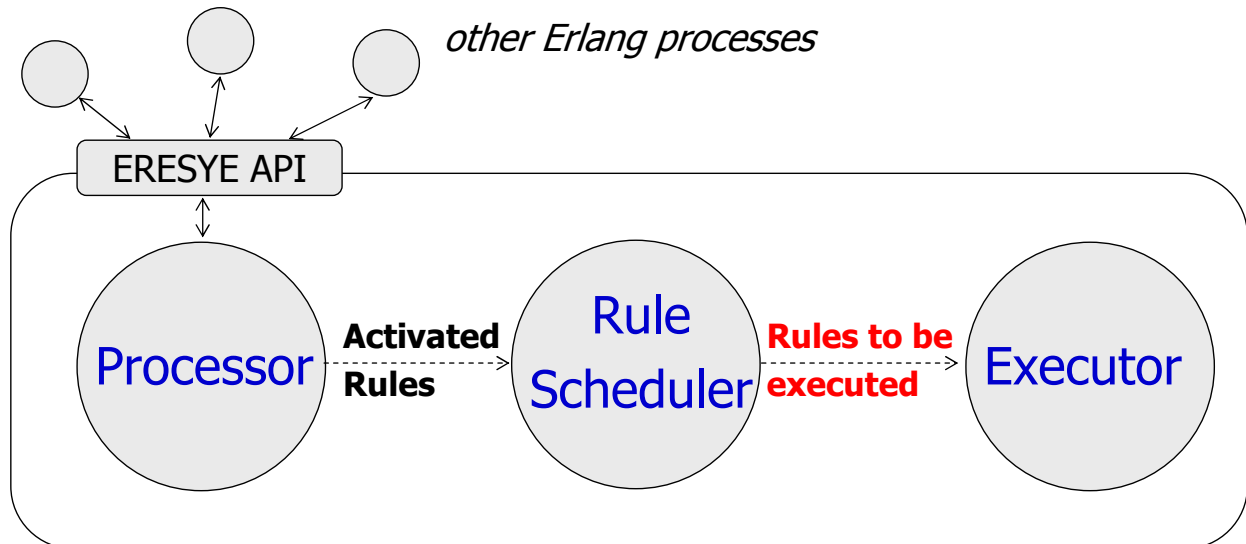


ERESYE Engine Architecture



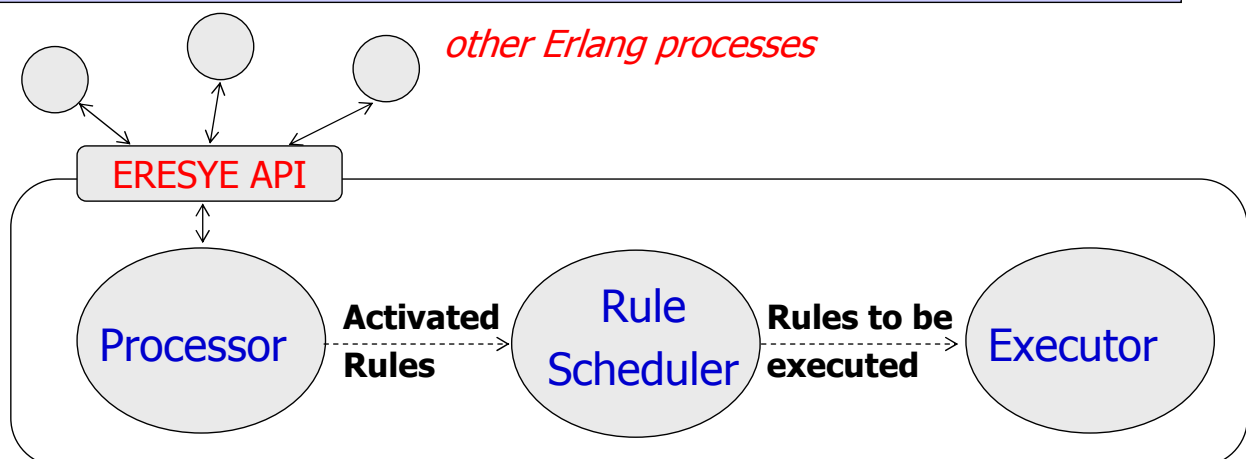
- The **Processor** executes the RETE algorithm
- Once a rule is activated, it is passed to the **Rule Scheduler** for execution

ERESYE Engine Architecture



- The **Rule Scheduler** selects an active rule according to its priority and sends it to the **Executor**
- The **Executor** concerely runs the function associated to the rule

ERESYE Engine Architecture

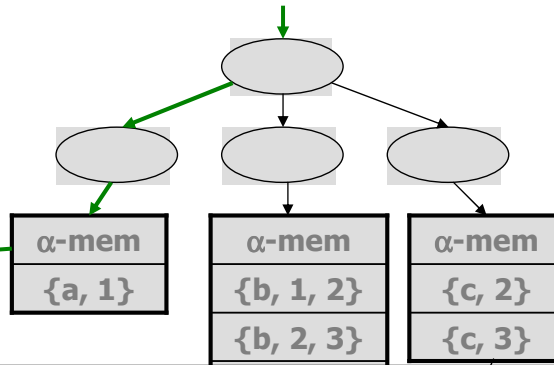


- Any Erlang process can interact with an engine through the API to
 - ✓ assert or retract a fact
 - ✓ wait for the assertion of a fact with a given pattern
 - ✓ add a new rule or remove an existing rule
 - ✓ deactivating/activating a rule
 - ✓ change rule priority, etc.

ERESYE and RETE



```
sample_rule(Engine,
  {a, X},
  {b, X, Y},
  {c, Y}) ->
```



β -mem (matches for C1)

- ERESYE exploits Erlang matching mechanism
- A "fun" is associated to each node to perform matching
- This "fun" is derived by analyzing the source code of the function defining the rule

{a, 1}	{b, 1, 2}
--------	-----------

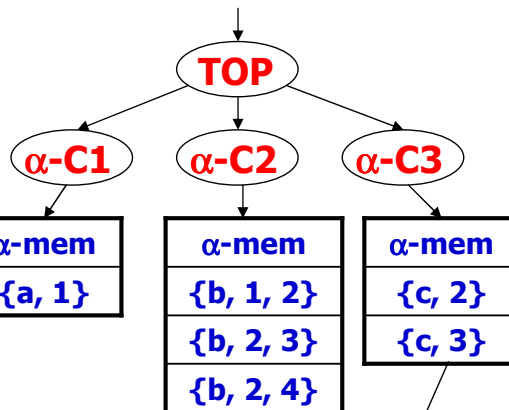
β -mem (matches for C1 and C2 and C3)		
(a 1)	{b, 1, 2}	{c, 3}

Rule Activation

ERESYE and RETE



```
sample_rule(Engine,
  {a, X},
  {b, X, Y},
  {c, Y}) ->
```



β -mem (matches for C1)

{a, 1}

Node	Fun
α -C1	fun ({a, X}) -> true end.
α -C2	fun ({b, X, Y}) -> true end.
α -C3	fun ({c, Y}) -> true end.

β -mem (ma
{a, 1}

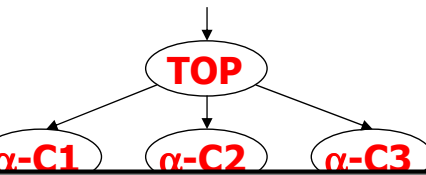
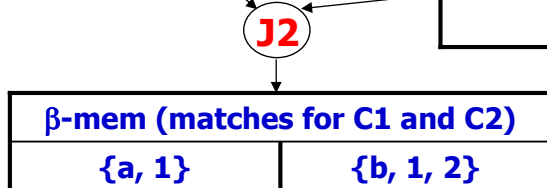
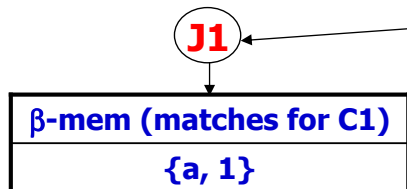
β -mem (C2 and C3)		
(a 1)	{b, 1, 2}	{c, 3}

Rule Activation

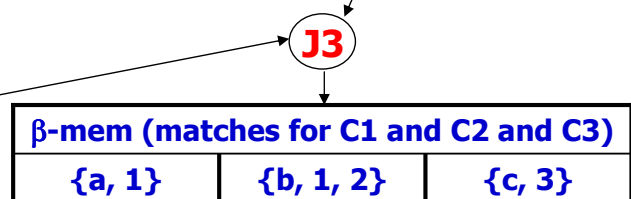
ERESYE and RETE



```
sample_rule(Engine,
            {a, X},
            {b, X, Y},
            {c, Y}) ->
```



Node	Fun
J1	fun ([], {a, X}) -> true end.
J2	fun ([{a, X}], {b, X, Y}) -> true end.
J3	fun ([{a, X}, {b, X, Y}], {c, Y}) -> true end.



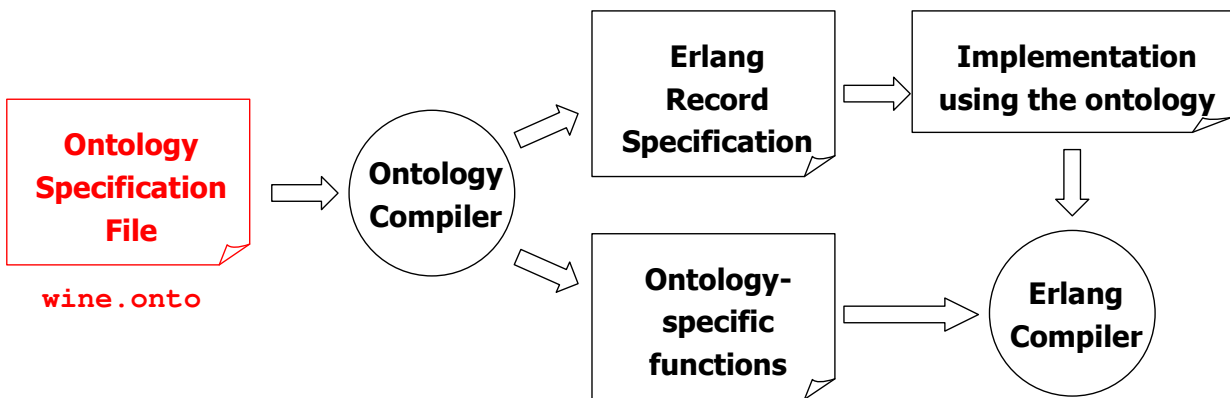
Rule Activation

Ontology Modeling in ERESYE



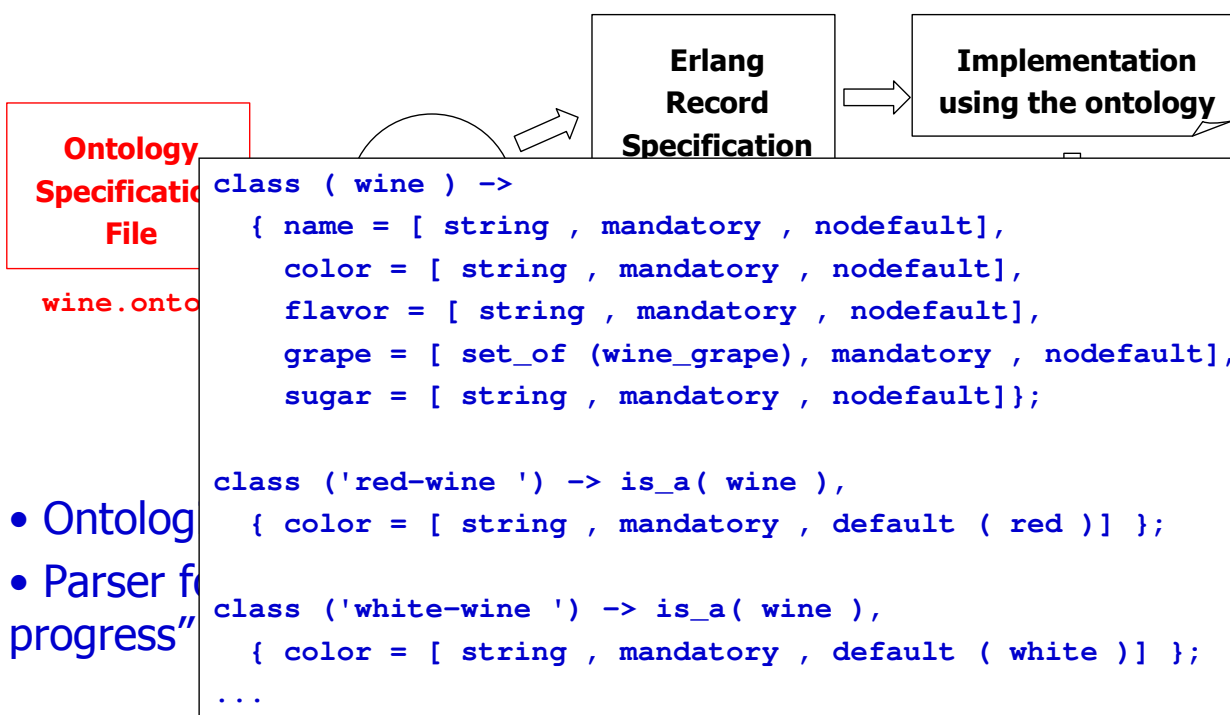
- In writing intelligent systems, it is usual to model the "universe of discourse" by using *ontologies*.
- ERESYE is able to support ontologies—i.e. *classes* with hierarchies and *objects*—and use them in engines.
- But Erlang is not object-oriented, so ERESYE comes with a tool—the ontology compiler—able to transform an object-based ontology specification into an Erlang readable (non-object-based) form, maintaining semantics.

Ontology Modeling in ERESYE



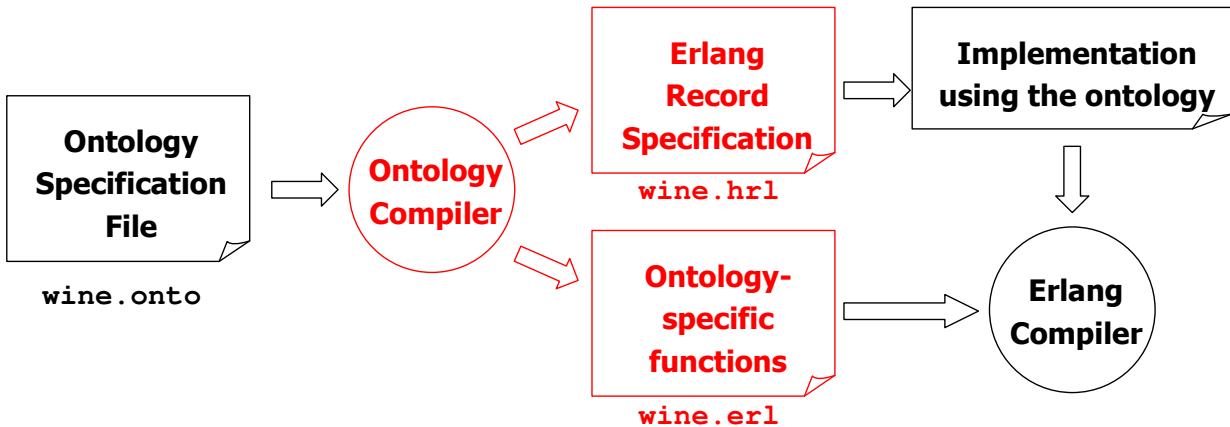
- Ontologies are written in Erlang-like notation
- Parser for standard languages (OWL or RDF) are a “work-in-progress”

Ontology Modeling in ERESYE



- Ontology
- Parser for
- progress”

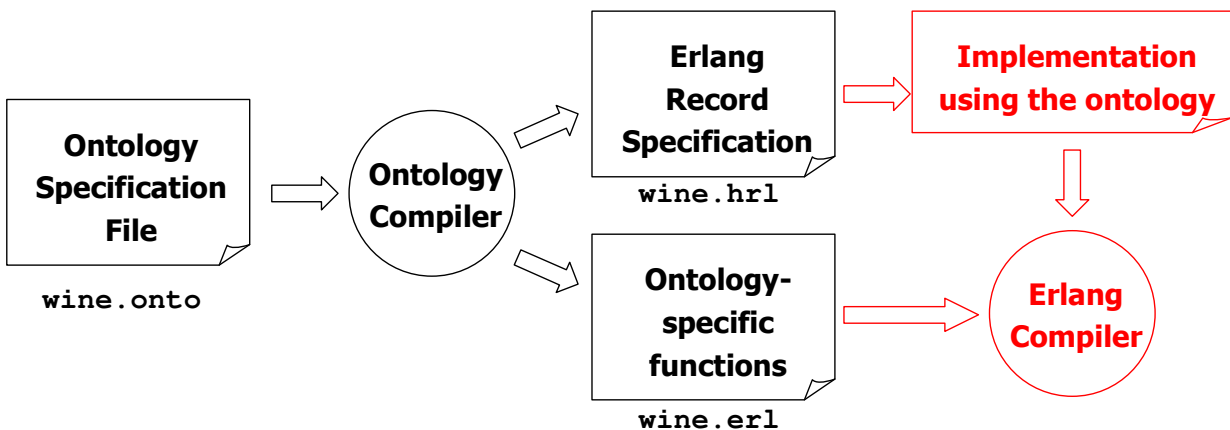
Ontology Modeling in ERESYE



- The Ontology Compiler (provided with ERESYE) parses the `.onto` file and generates two files:

- `.hrl`, classes are transformed into Erlang records, hierarchy is flattened
- `.erl`, functions that maintain a track of hierarchies ("is_a" functions) and perform typecasting (*see the details in the paper*)

Ontology Modeling in ERESYE



- The generated files can be directly used in the implementation
- Rules can directly use generated records

CLIPS/ERESYE Comparison



- Knowledge Representation:

- CLIPS

- Symbols
 - Simple facts
 - Objects with hierarchies

- ERESYE

- Symbols (provided by Erlang)
 - Facts modeled with tuples (provided by Erlang)
 - Object and hierarchies (supported by the ERESYE ontology compiler)

CLIPS/ERESYE Comparison



- Rule expression

- CLIPS

- first-order logic predicates with compound variable bindings
 - guards with logic connectives (and, or, not)
 - guards with built-in and user-defined functions

- ERESYE

- Function heads are logic predicated with also compound variable bindings
 - Guards are supported as well
 - Guards can use BIFs, but cannot use user-defined functions (are they really necessary? Often BIFs suffice!)

CLIPS/ERESYE Comparison



- Integration with other systems

- CLIPS

- It is C-based
 - Integration is performed by statically linking CLIPS interpreted with the target system
 - The interpreter source code is always necessary

- ERESYE

- Automatically integrated if the target system is written in Erlang
 - Otherwise you may use ports, drivers, jinterface, etc.

Conclusions and Future Work



- Once, a person in this room said "Erlang is perfect for AI", ERESYE is the demonstration!
- It opens new and interesting scenarios for the Erlang languages (e.g. robotic application?)
- ERESYE is now a part of eXAT, an Erlang-based agent programming platform developed at the Univ. of Catania (it will be presented at EUC'05)
- But some work is still needed:
 - Evaluate the performance and optimize the algorithm
 - Improve the "not syntax" (the syntax to express that a fact **has not** to be asserted in the KB for a rule to be triggered)
 - Provide a means to use standard ontology descriptions
 - Improve ontology handling
 - Any other suggestion from audience?

Thank you!



Erlang Training and Consulting Ltd
www.erlang-consulting.com

Contact Info

Francesca Gangemi, EMail: francesca@erlang-consulting.com

Corrado Santoro, EMail: csanto@diit.unict.it