



www.ocopomo.org

Open
COllaboration for
POlICY MOdelling



DRAMS

A Declarative Rule-based Agent Modelling System

ECMS Conference, Kraków (PL), 8th June 2011

Ulf Lotzmann

ulf@uni-koblenz.de



- ❖ Overview
 - What is a Declarative Rule Engine?
 - What it is useful for?
- ❖ Components and integration
 - How can Rule Engines used with Agent-based Systems?
- ❖ DRAMS Design
 - What are the major design aspects of DRAMS?
- ❖ DRAMS User interface
 - How is DRAMS used to create and run simulation models?
 - How could the modelling process supported further?
- ❖ Outlook
 - What are the main objectives for future work?

Overview: Declarative Rule Engine (1)



- ❖ Rule Engine is a software system, consisting of:
 - A fact base, which stores information about the state of the world in the form of facts.
 - A rule base, which stores rules representing knowledge how to process certain facts stored in fact bases. A rule consists of a condition part (called left-hand side, LHS) and an action part (called right-hand side, RHS).
 - An inference engine, which controls the inference process by selecting and processing the rules which can fire on the basis of certain conditions, in order to draw conclusions from existing facts.

Overview: Declarative Rule Engine (2)



❖ Declarative Rule Engine

- Representation of rules quite close to natural language descriptions
- “More realistic” representation of human approaches to reasoning and problem solving

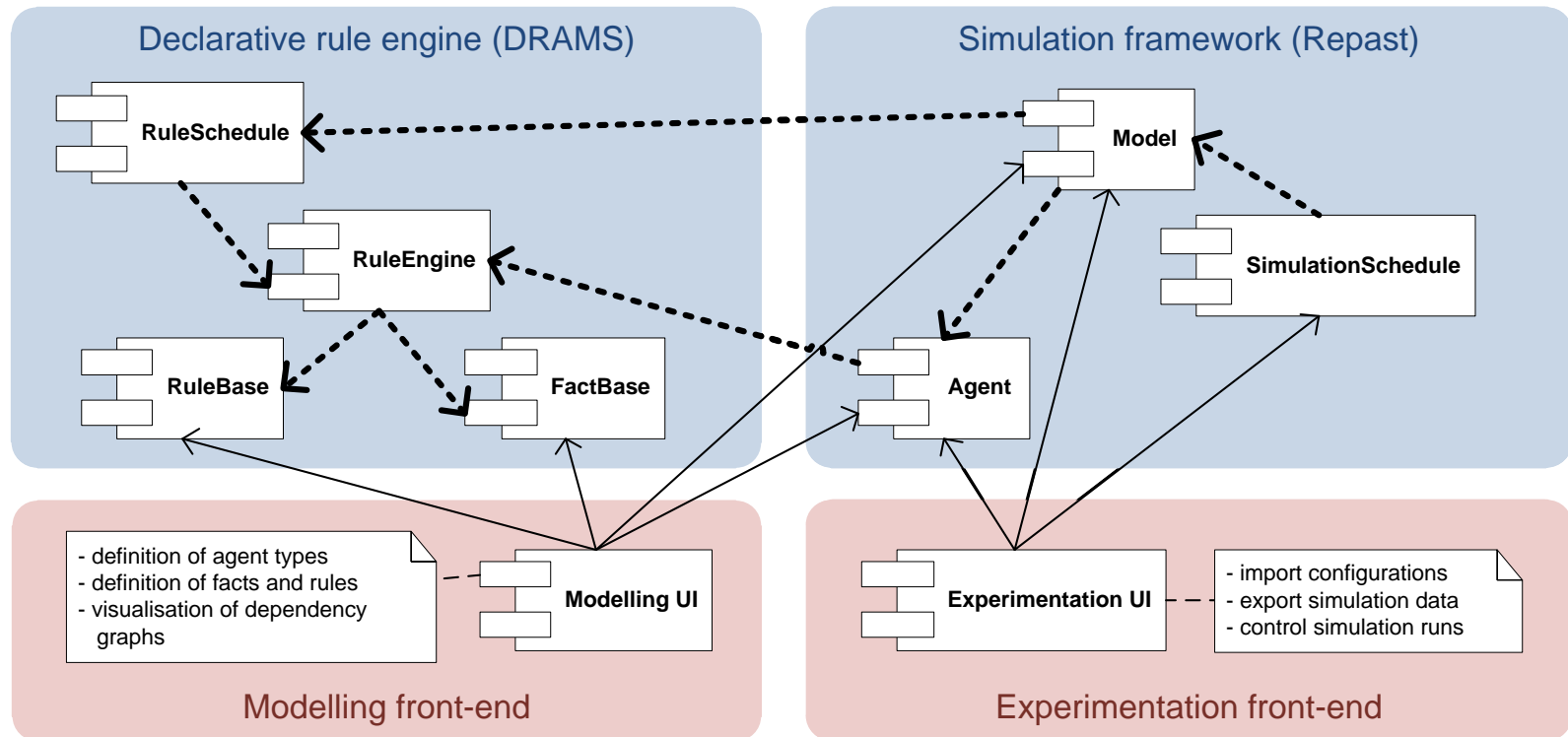
❖ Rule engines and agent-based models

- Modelling agent reasoning via expert system
- Other agent functionality with imperative code

❖ Problems with existing software products

- Optimisation for static fact bases causes performance bottleneck with dynamic simulation data
- Restricted agent autonomy due to shared rule engine

Components and integration



❖ Basic idea

- Equip agents with expert system capabilities: describing agent behaviour by declarative rules
- Individual rule sets for each agent type (or even instance)
- Individual working memory for each agent instance

❖ Rule engine component for multi-agent simulation models

- Distributed rule engine: agents behave in simulation runs autonomously according to their rule specifications
- Simulation dynamics is generated by individual agent behaviour, together with interaction between agents (inter-agent communication)

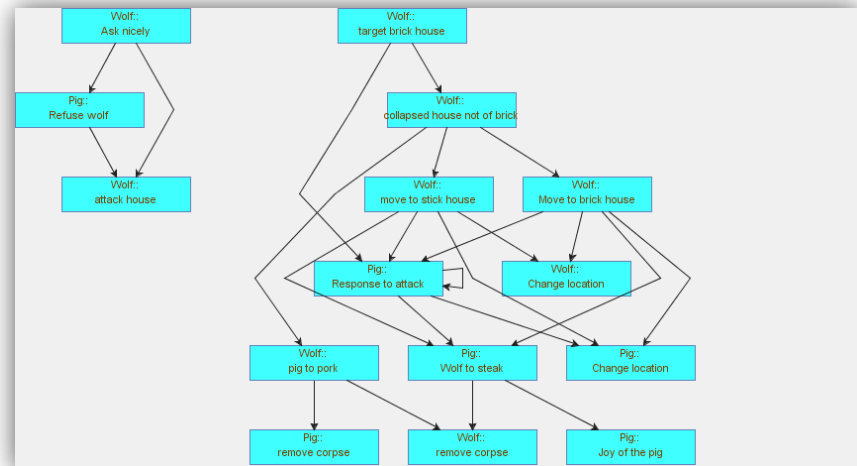
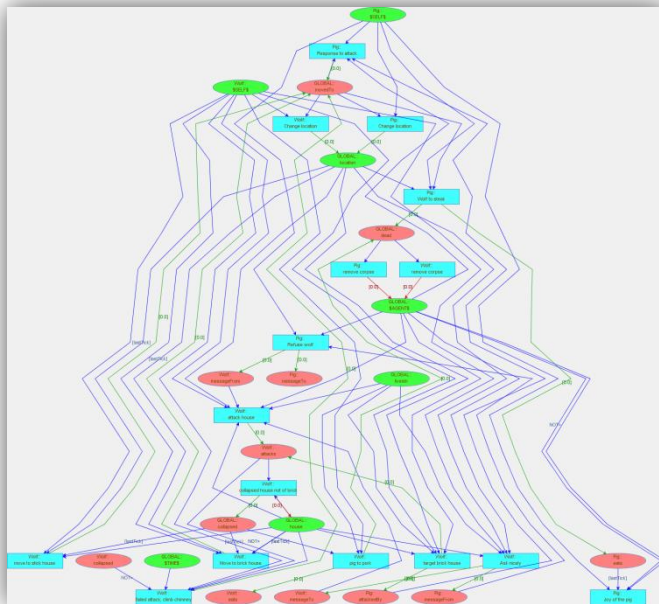
❖ Distributed rule engines

- Arbitrary number of agent types with type-specific rule bases and initial fact base configurations
- Arbitrary number of instances (objects) for each agent type with individual fact bases
- One global rule engine, containing “world knowledge”

❖ Communication between agents

- Reading and writing access to the shared global fact base
- (Reading and) writing access to other agents' fact bases

- ❖ Forward-chaining inference engine
- ❖ Incorporates a data-driven rule scheduling mechanism
 - Efficiently cope with intensely dynamic fact base contents
 - Based on dependency graphs



ОСОБОМО



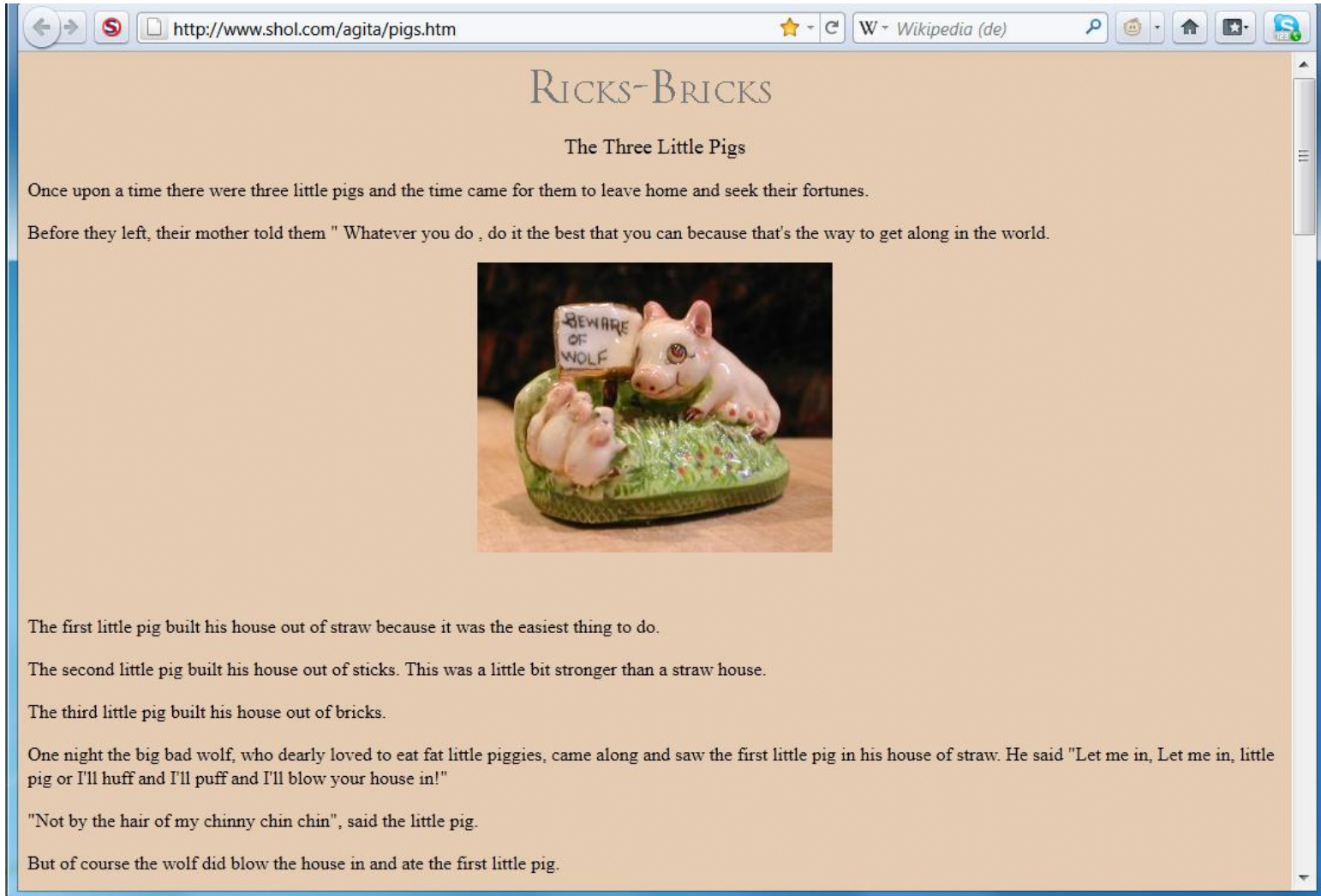
❖ Access to rule engines:

- Instantiation of Fact or Rule classes within the tool source code
- Reading/writing/editing of configuration files using a specific language (OPS5 style, similar to JESS)
- Reading/writing of XML-based configuration files
- Interactively defining facts and rules with a GUI

❖ Supporting the modelling process

- Calculation and displaying of dependency graphs

DRAMS UI: example model

A screenshot of a web browser window. The address bar shows 'http://www.shol.com/agita/pigs.htm'. The page title is 'RICKS-BRICKS' and the subtitle is 'The Three Little Pigs'. The text of the story is displayed in a serif font. In the center, there is a photograph of a ceramic pig figurine sitting on a green base, holding a sign that says 'BEWARE OF WOLF'. The browser's toolbar includes back, forward, and search buttons, as well as a search bar and a 'Wikipedia (de)' tab.


http://www.shol.com/agita/pigs.htm

RICKS-BRICKS

The Three Little Pigs

Once upon a time there were three little pigs and the time came for them to leave home and seek their fortunes.

Before they left, their mother told them " Whatever you do , do it the best that you can because that's the way to get along in the world.



The first little pig built his house out of straw because it was the easiest thing to do.

The second little pig built his house out of sticks. This was a little bit stronger than a straw house.

The third little pig built his house out of bricks.

One night the big bad wolf, who dearly loved to eat fat little piggies, came along and saw the first little pig in his house of straw. He said "Let me in, Let me in, little pig or I'll huff and I'll puff and I'll blow your house in!"

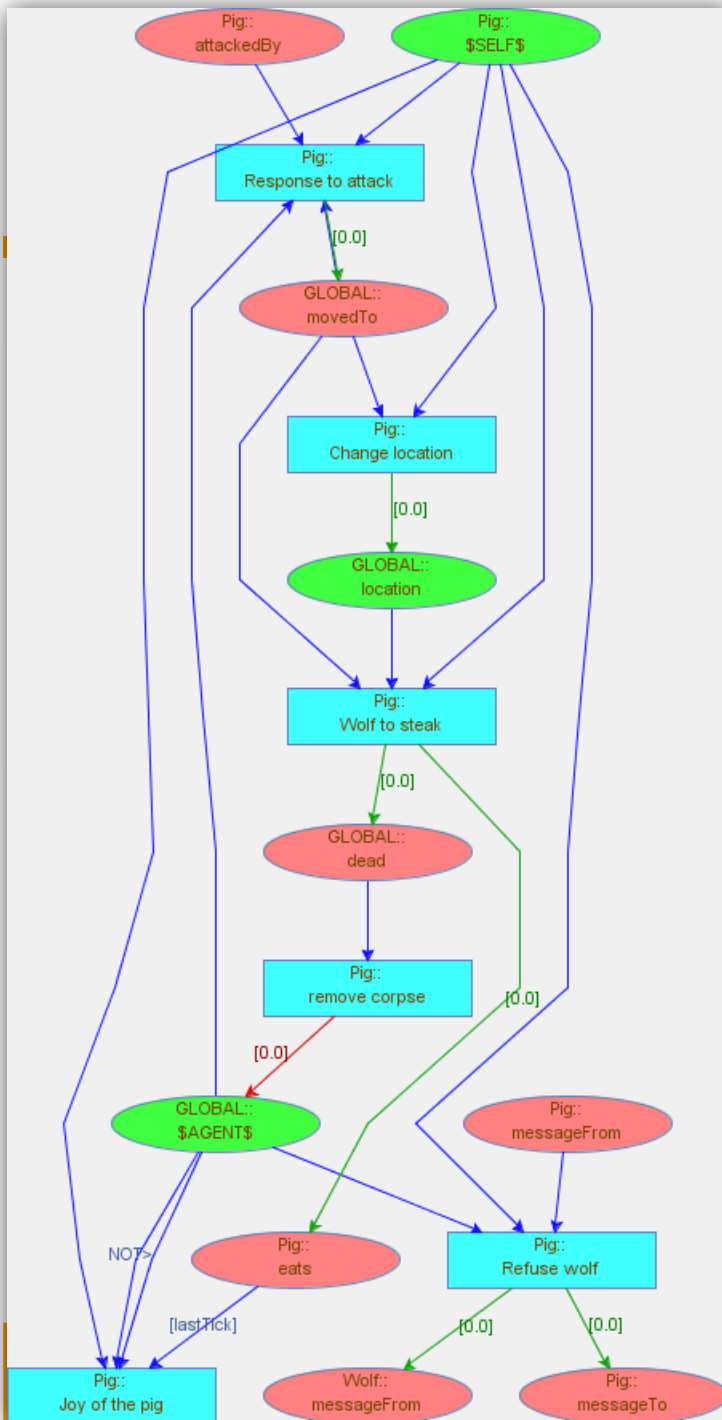
"Not by the hair of my chinny chin chin", said the little pig.

But of course the wolf did blow the house in and ate the first little pig.

When the wolf finally found the hole in the chimney he crawled down and **KERSPLASH** right into that kettle of water and that was the end of his [the third little pigs] troubles with the big bad wolf.

```
(defrule Pig::"Wolf to steak"
  (global::location (place "fireplace") (object ?wolf) )
  (global::movedTo (object "kettle") (place "fireplace") )
  ($SELF$ (name ?name) )
=>
  (assert (global::dead (agent ?wolf) ) )
  (assert (eats (dinner ?wolf) ) )
  (print "?wolf and the kettle have met at the fireplace")
  (print "?wolf is boiled to death in the kettle")
  (print "?name enjoys a wolf dinner")
)
```

```
rule = new Rule("Wolf to steak");
rule.addLHSClause(new RetrieveClause("(global::location (object ?wolf) (place \"fireplace\") )"));
rule.addLHSClause(new RetrieveClause("(global::movedTo (object \"kettle\") (place \"fireplace\") )"));
rule.addLHSClause(new RetrieveClause("( $SELF$ (name ?name) )"));
rule.addRHSClause(new ActionClauseAssert("(assert (global::dead (agent ?wolf) ) )"));
rule.addRHSClause(new ActionClauseAssert("(assert (eats (dinner ?wolf) ) )"));
rule.addRHSClause(new JavaActionClause("(print \"?wolf and the kettle have met at the fireplace\" )"));
rule.addRHSClause(new JavaActionClause("(print \"?wolf is boiled to death in the kettle\" )"));
rule.addRHSClause(new JavaActionClause("(print \"?name enjoys a wolf dinner\" )"));
rb.addRule(rule);
```



(defrule Pig::"Wolf to steak"

(global::location (place "fireplace") (object ?wolf))
 (global::movedTo (object "kettle") (place "fireplace"))
 (\$SELF\$ (name ?name))

=>

(assert (global::dead (agent ?wolf)))
 (assert (eats (dinner ?wolf)))
 (print "?wolf and the kettle have met at the fireplace")
 (print "?wolf is boiled to death in the kettle")
 (print "?name enjoys a wolf dinner")
)

DRAMS UI: experimentation



```
***** SCHEDULE AT TIME 2.0*****  
----- TASKS FOR TIME 2.0 -----  
---> TASK NO 5  
[...]  
* ACTION ActionRuleFire FOR RULE 'Wolf::attack house'  
  REASON:  
    NEW FACTS AVAILABLE FOR: GLOBAL::location  
    INSTANCES: Wolf (1);  
* ACTION ActionRuleFire FOR RULE 'Pig::Change location'  
  REASON:  
    NEW FACTS AVAILABLE FOR: GLOBAL::movedTo  
    INSTANCES: Pig (1);  
* ACTION ActionRuleFire FOR RULE 'Pig::Wolf to steak'  
  REASON:  
    NEW FACTS AVAILABLE FOR: GLOBAL::movedTo; GLOBAL::location  
    INSTANCES: Pig (1);  
[...]  
----- TASKS FOR TIME 3.0 -----  
---> TASK NO 0  
[...]
```

DRAMS UI: experimentation



*** 2.0 ***

[bigBadWolf.Move to brick house] "bigBadWolf ready to move"
[bigBadWolf.Move to brick house] "bigBadWolf has moved from the stick house to the brick house"
[bigBadWolf.Change location] "bigBadWolf is moving to house-2"
[bigBadWolf.Ask nicely] "bigBadWolf has asked pig-2 if he can enter his house of brick"
[bigBadWolf.target brick house] "bigBadWolf attacks the house of brick"
[bigBadWolf.failed attack; climb chimney] "bigBadWolf has attacked the house of brick, house-2"
[bigBadWolf.failed attack; climb chimney] "but house-2 has not collapsed!!"
[pig-2.Refuse wolf] "pig-2 has replied: Not by the hair on my chinny-chin-chin"
[bigBadWolf.Change location] "bigBadWolf is moving to fireplace"
[bigBadWolf.Change location] "bigBadWolf is moving to house-2"
[pig-2.Response to attack] "bigBadWolf is a Wolf moving to the fireplace"
[pig-2.Response to attack] "pig-2 is moving the kettle to the fireplace"
[pig-2.Wolf to steak] "bigBadWolf and the kettle have met at the fireplace"
[pig-2.Wolf to steak] "bigBadWolf is boiled to death in the kettle"
[pig-2.Wolf to steak] "pig-2 enjoys a wolf dinner"

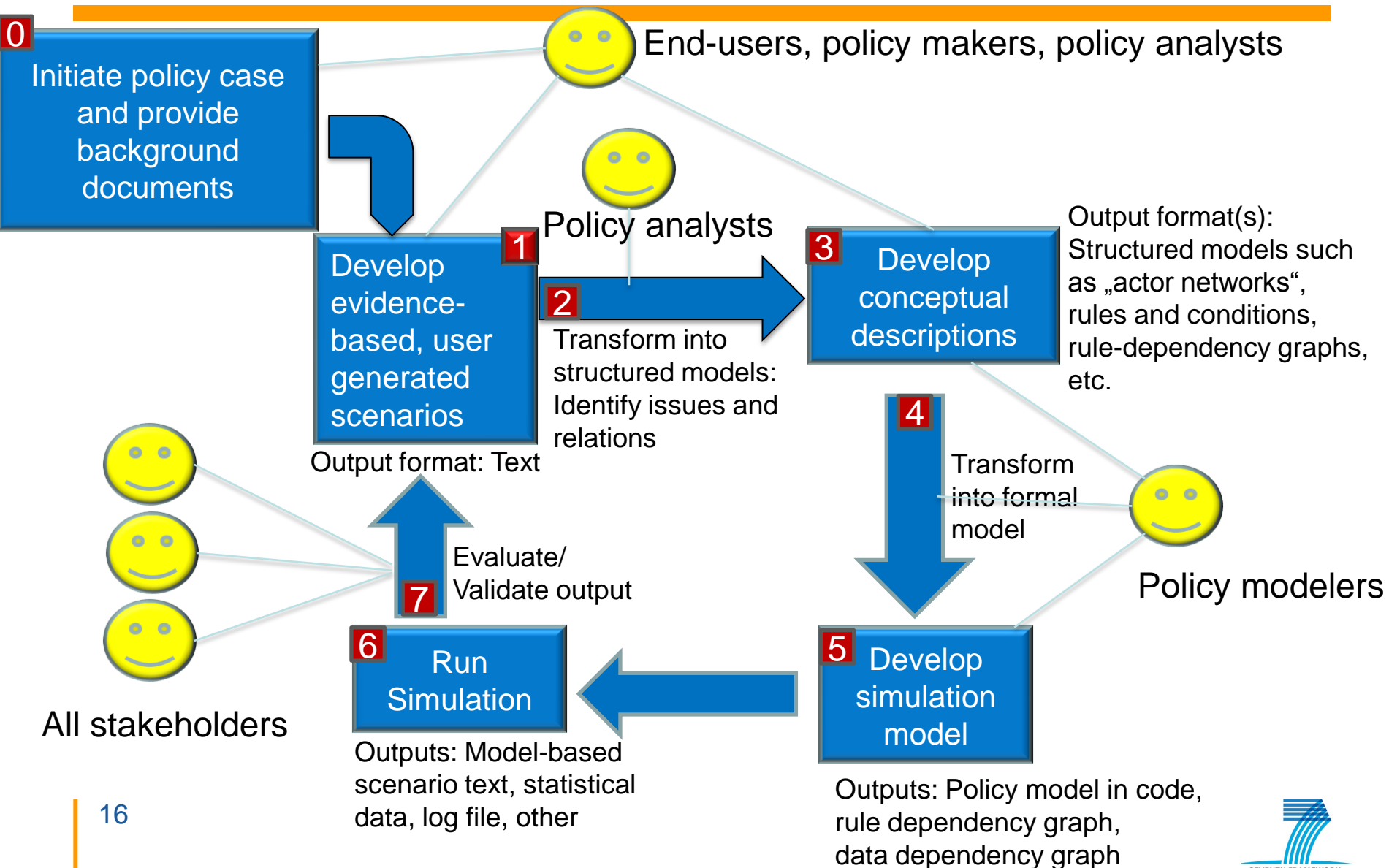
Time to run: 0.486 seconds

*** 3.0 ***

[pig-2.Joy of the pig] "pig-2 dances for joy because no wolves remain alive"

Time to run: 0.026 seconds

DRAMS UI: OCOPOMO policy development process



❖ DRAMS → DREAMS

- New functionality: plugin interface, meta rules, optimisation
- Java-based implementation: integrable with widely applied simulation tools (e.g. Repast)
- Completely open source (including external libraries)

❖ OCOPOMO Toolbox

- CMS system (Alfresco)
- Several components: integrated (viewer, user and data management) or linked via CMS data repository:
 - Text annotation tool (Eclipse plugin)
 - CCD tool (Eclipse plugin)
 - DREAMS modeller (Eclipse plugin)
 - Simulation tool (Repast)



www.ocopomo.org

Open
COllaboration for
POlICY MOdelling



Many thanks for your attention!

Project partners:



KSR



REGIONE CAMPANIA