Agent-based Dialogue Management

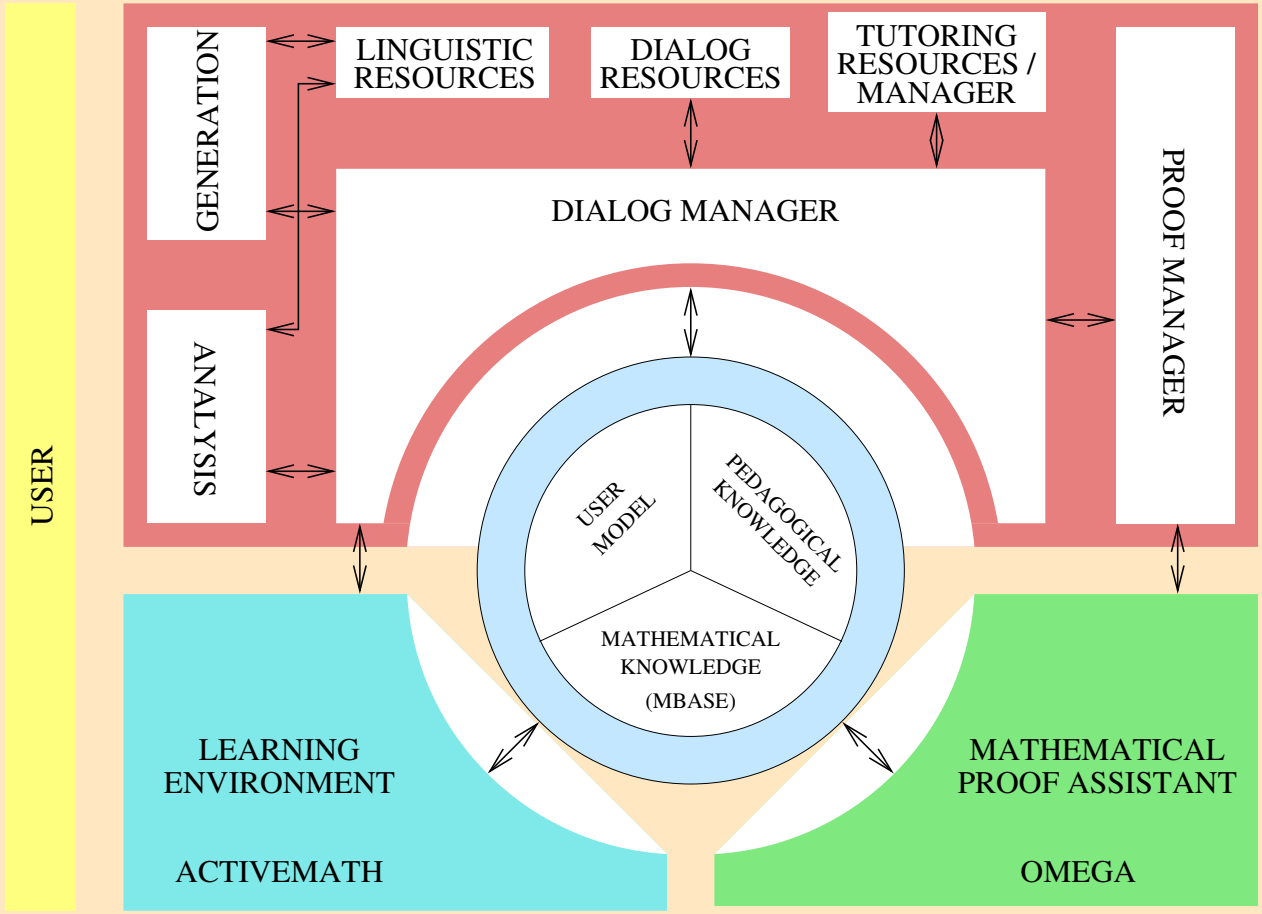
Mark Buckley

markb@ags.uni-sb.de

IGK Annual Meeting 2005

July 9th, 2005

The DIALOG Project



Flexible natural language dialogue in a mathematical tutorial system

The Dialogue Manager

Function:

- Handles student/system interaction
- Intercommunication of system modules
- Controls top-level execution

The Dialogue Manager

Function:

- Handles student/system interaction
- Intercommunication of system modules
- Controls top-level execution

Uses the Information State Update approach

- Information State
- Dialogue moves
- Update rules
- Update strategy

DIALOG Demonstrator

- Focus on one dialogue from the corpus
- Used ISU dialogue manager built using Rubin

DIALOG Demonstrator

- Focus on one dialogue from the corpus
- Used ISU dialogue manager built using Rubin

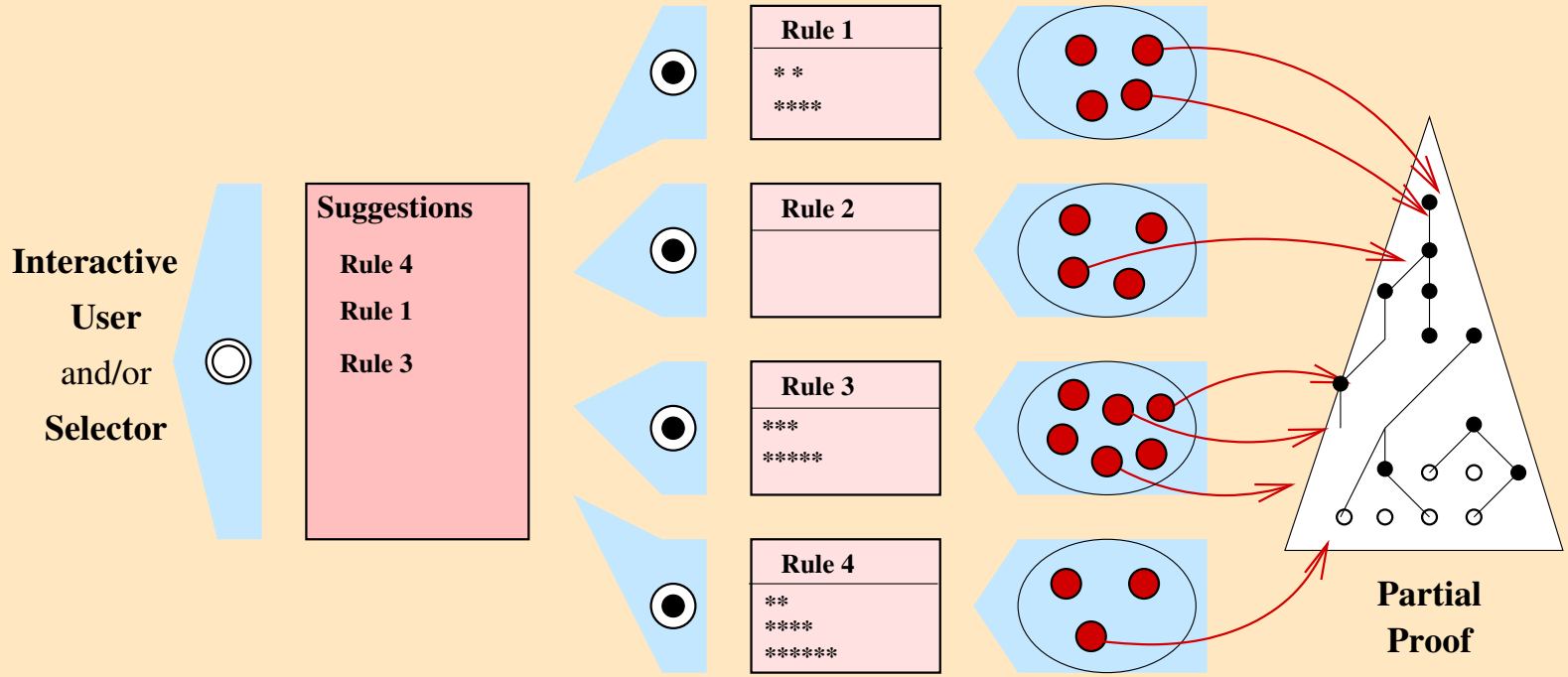
Desiderata:

- Direct access to IS
- Better control of information flow
- Runtime flexibility (e.g. redefinability)
- A meta-level for choice of IS update

Ω-Ants

A suggestion mechanism for interactive proof planning in ΩMEGA

- Concurrent agents represent ΩMEGA proof planning commands



Ω-Ants

Ω-Ants binds in external systems

- The command

$$\frac{Premlist}{Conc}\,Otter$$

  is represented by an Ω-Ants agent

Ω-Ants

Ω-Ants binds in external systems

- The command

$$\frac{Premlist}{Conc}\,Otter$$

  is represented by an Ω-Ants agent

Benefits of Ω-Ants:

- Concurrent agent execution
- Runtime flexibility
- Resource adaptiveness

An Agent-based Dialogue Manager

- Build a platform for dialogue management
- Use agent-based techniques from Ω-Ants

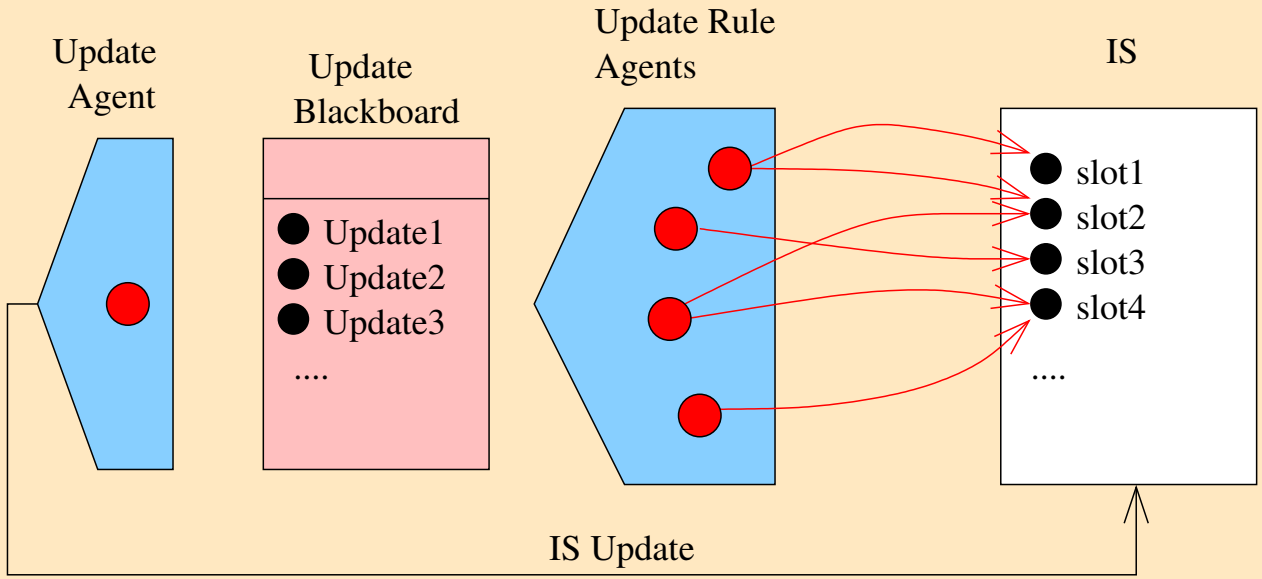
An Agent-based Dialogue Manager

- Build a platform for dialogue management
- Use agent-based techniques from Ω-Ants

It will provide:

- Support for ISU approach
- Easy integration of external modules
- Concurrent execution by software agents
- Hierarchical design

Architecture



Defining an IS

- A set of named, typed slots, possibly with an initial value
- E.g.:

```
(tutorialmode :doc "the current tutorial mode"
              :init "min"
              :test #'(lambda (val list) (find val list :test #'equal))
              :args (("min" "soc" "did")))
```

- `tutorialmode` can now be accessed by update rules and written by the update agent

Defining Update Rules

- Computes updates of IS slots
- Consists of preconditions, sideconditions, effects:

$$\frac{p_1 : test_1, \ \ldots \ , p_n : test_n}{(e_1, expr_1), \ \ldots \ , (e_m, expr_m)} U \ ((s_1, expr_1), \ \ldots \ , (s_l, expr_l))$$

- Fires when the current IS satisfies the preconditions

Defining Update Rules


```
(ur~define-update-rule :name "NL Analyser"
                       :preconds ((utterance :test #'stringp))
                       :sideconds ((result (call-to-NL-module utterance))
                                   )
                       :effects ((lm result))
                       )
```

- Preconditions are freely definable tests on IS slots
- Sideconditions can call arbitrary functions
  - Interface to external modules
- Set of evaluated effects constitutes an IS update

Benefits of new Dialogue Manager

What do we gain from this approach?

- Benefits from Ω-Ants
  - Concurrency, flexibility, resource adaptiveness
- A natural way to integrate external systems
- Application of heuristics in update strategy
  - Dialogue Manager can control top-level system execution
  - Don't have to rely on rule ordering
- Better interleaving of NLU and Proof Management