A Logic Based Language for Engineering Agent Applications

Keith L Clark
Imperial College
University of Queensland
(joint work with Peter J Robinson, UQ)

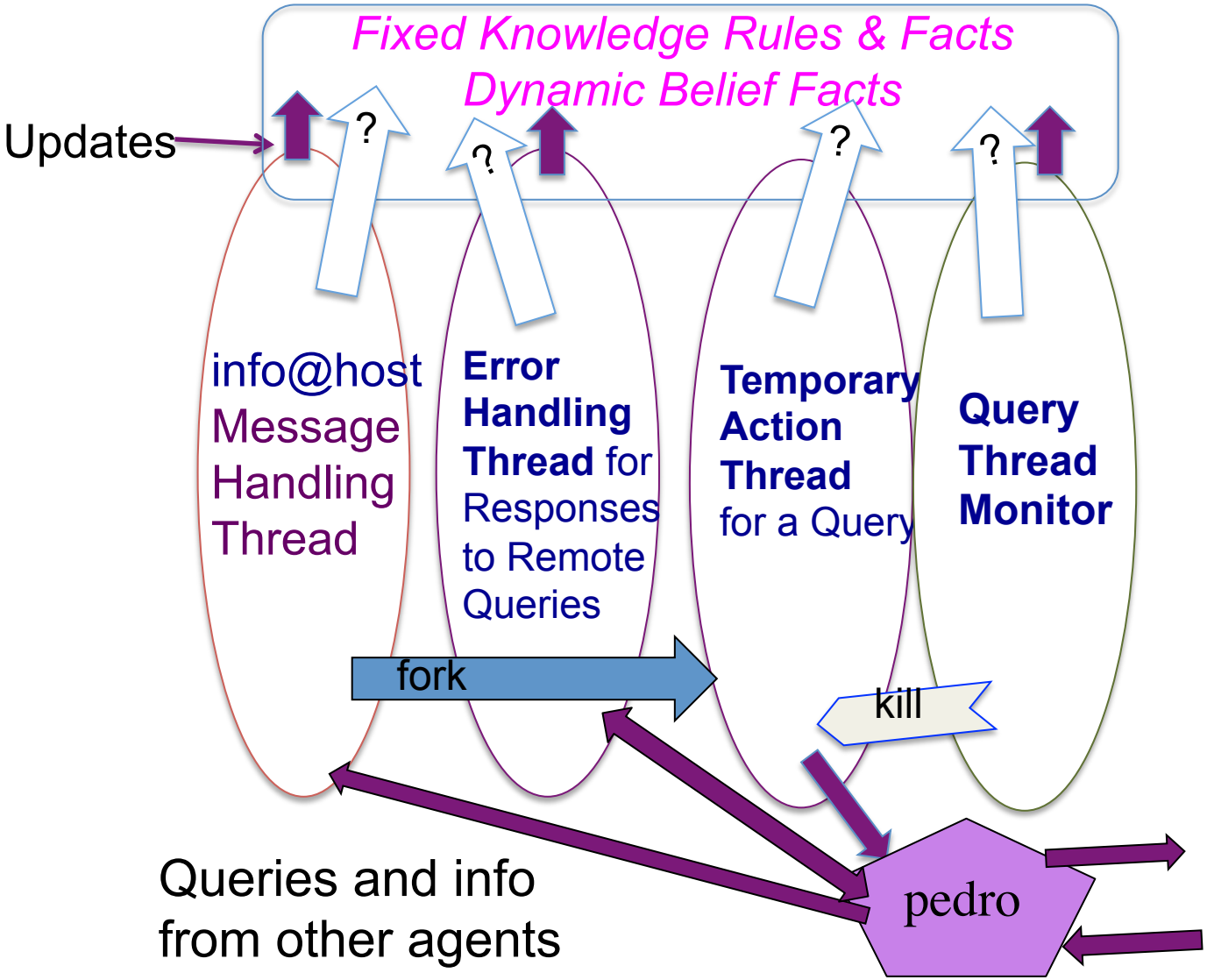
Key features of QuLog

- **Dynamic relations** – an agent's current beliefs
  - Defined solely by updatable facts
- **Static Relations** – an agent's knowledge
  - Conditional rules, committed choice rules, and non-updatable facts.
  - More declarative and concise than Prolog – closer to predicate logic
    - Evaluable expressions as call arguments
    - Including list and set comprehension expressions
  - Non-deterministic pattern matching support for string + list processing
- **Functions** – syntactic sugar for functional relations
  - Committed choice rewrite rules
  - Relation queries as commitment tests
  - All functions must be named – no lets or lambdas
  - Strict evaluation
- **Actions** – an agent's behavioral knowledge
  - Committed choice action rules
  - Dynamic fact updates, thread forking, I/O and message communication
  - Actions may call functions and relations but not vice versa

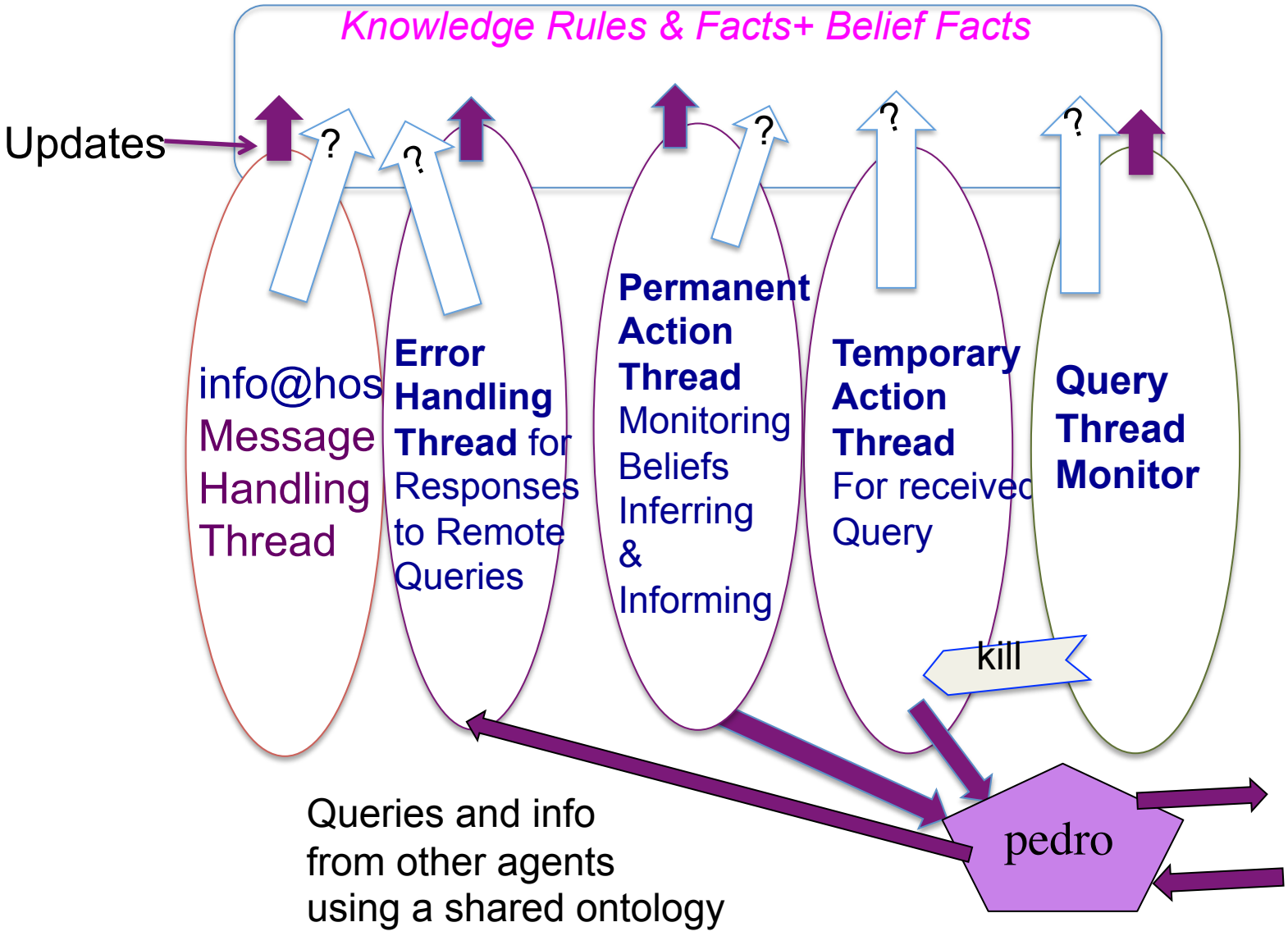
Key features continued

- **Multi-threaded** - threads execute action calls
- **Type declarations** - required for functions
- **Moded type declarations -** required for relations and actions
- **Modes specify** which arguments:
  - **must be ground -** the default mode optional ! annotation
  - **will be ground** after a successful call - ? annotation
  - **may not be ground** after a successful call - ?? annotation
  - **will be unchanged** - @ annotation
  - **Hybrid modes** e.g. list(?nat)
- **New types can be defined**. Union types allowed.
  - Primitive type atomic is union of nat, int, num, atom, string
- **Run-time test for any type**
- **Enforced type and mode safe** for meta-level and remote querying
- **Function call arguments** must *all* be ground at time of the call

QuLog Information Agent



QuLog Pro-active Reasoning Agent



QuLog Agents

- **Agents** are named *multi-threaded QuLog processes*
- Names are registered with a Pedro communication server
- One thread is designated default message handler
- Threads share the Beliefs and Knowledge - no sharing of variables
- **Internal thread co-ordination**
  - using atomic updates of beliefs and suspendable queries to the beliefs
  - Internal messages
- **Agent co-ordination** via either:
  - Addressed messages routed via the Pedro server
  - Notifications sent to the Pedro server forwarded to agents that have lodged a covering subscription

The Glue - Pedro Communication Server

- Primarily a **pub/sub server**
- Agents publish *notifications* – as QuLog terms
- Routed to *all* other agents with a *current* Pedro *subscription* that *covers* the notification
- The **dynamic subscriptions** have the form:
  NTrmPtn :: Conds
- A subscription covers a notification term NTrm if
  NTrmPtn = NTrm & Conds holds
- Pedro *remembers* subscriptions *but not* notifications
  - a content based *router* **not** a *blackboard*
- Can handle 10,000 notifications per second with 1000 subscriptions

Dynamic Relations

dyn $r(t_1,..,t_k)$

$r(a_1,..,a_k)$ % all ground facts

....

Static Relations

rel $s(mt_1,..,mt_k)$ % moded type expressions

$s(a_1,..,a_k)$ % any term arguments

$s(a_1,..,a_k)$ <= *Conds* % uncommitted clauses

$s(a_1,..,a_k)$ :: *Test* % committed clauses

$s(a_1,..,a_k)$ :: *Test* <= *Conds*

Functions

fun $f(t_1,..,t_k)$ -> t

$f(a_1,..,a_k)$ -> *Exp* % all committed rules

$f(a_1,..,a_k)$ :: *Test* -> *Exp*

Actions

act $a(mt_1,..,mt_k)$ % moded type expressions

$a(a_1,..,a_k)$ % all committed rules

$a(a_1,..,a_k)$

$a(a_1,..,a_k)$ :: *Test* ~> Act_1 ; .... ; Act_k

Negated and quantified conditional tests

- Negated tests: not exists *EVars Predication*

  person(P) & not exists C,A (child_of(C,P) & age(C,A) & A>18)

- Quantified conditional tests:
  forall *AVars* (*EVars1 Conj1* => exists *EVars2 Conj2*)

  forall C,A (child_of(C,P) & age(C,A) => not A > 18)

or stronger condition:

  forall C,A (child_of(C,P) => exists A age(C,A) & not A > 18)

Set and list comprehension expressions

{Vars :: exists EVars Conj} [Vars :: exists EVars Conj]

e.g. { A :: exists C age_of(C,A) & A>17}

Set of ages over 17 of recorded children, no duplicates

[A :: exists C age_of(C,A) & A>17]

List of ages over 17 of recorded children, poss. duplicates

Set and list expressions *must have* ground values

Remote Queries

r(.....) <= C_1 &
(TypedVars :: Conds) query_to Ag &
C_2 &
...

Action handing of received remote query
in message handling thread of Ag

Query query_from Client;
respond_remote_query(Query, Client)

Watch debugging

- No step by step query tracing
- Instead a watch can be placed on any number of relations, functions and actions
- Log displayed each time a watched def. is used
- Invisible writes are inserted in the rules

Source

http://staff.itee.uq.edu.au/pjr/HomePages/QulogFiles/qulog0.9.tgz