User-Oriented Policy Management

Ian Wakeman **David Weir** **Bill Keller**

Julie Weeds **Tim Owen** **Jon Rimmer**

Jon Robinson **Thom Heslop**

Dept of Informatics, University of Sussex

November 30, 2004

1 Overview

- The anatomy of a user-oriented computing context
  - Description Logic
  - Implementing policies in description logic

2 Motivation

The Pervasive Computing vision:

- Lots of devices, sensors, actuators, gadgets - in the home, car, workplace, street, clothes...
- Networked and accessible. Sending out events, and handling requests. Software services form the layer above:
  - Heating Service (manages all the temp sensors, thermostats, valves, vents etc.)
  - Location Service (RFID tags, motion sensors)

3 Research Questions

- Lots of current research in:
  - The devices themselves (hardware, comms...)
  - Middleware, infrastructure, platforms
  - Web Services, XML, DAML, Zero-Conf
  - Software Agents, Semantic Web, AI Planning
- In a world of pervasive computing, with a rich set of services, we want to make use of it all
- We'd like to configure and compose services

3.1 Programmer or User?

- Much of the research is software-centric:
  WSDL, DAML, Jini etc, allow programs to tie services together and configure their behaviour
- What about the users?
- Pervasive computing is supposed to be for everyday folk, i.e. non-technical With a heating service and location service, how will I actually get them to do what I want?
  - Switch off the heating when nobody is at home

3.1.1 Policies

- When users have pervasive computing, they can use it to tailor their environment:
  - When the house is empty for more than 30mins, turn the heating down
  - Before 7pm, if my mobile rings when I'm at home, re-direct the call to my landline instead
  - Email me when Ian is back in his office
  - Print short documents on the nearest printer
- These rules or preferences, we call policies

3.2 Bridging the User-System Gap

- Users, especially non-technical ones, prefer using speech, text and GUIs, not writing code
- To implement policies, we need to convert them into a formal representation, suitable for use by software that connects services
- Our project, Natural Habitat, seeks to explore how feasible this is...

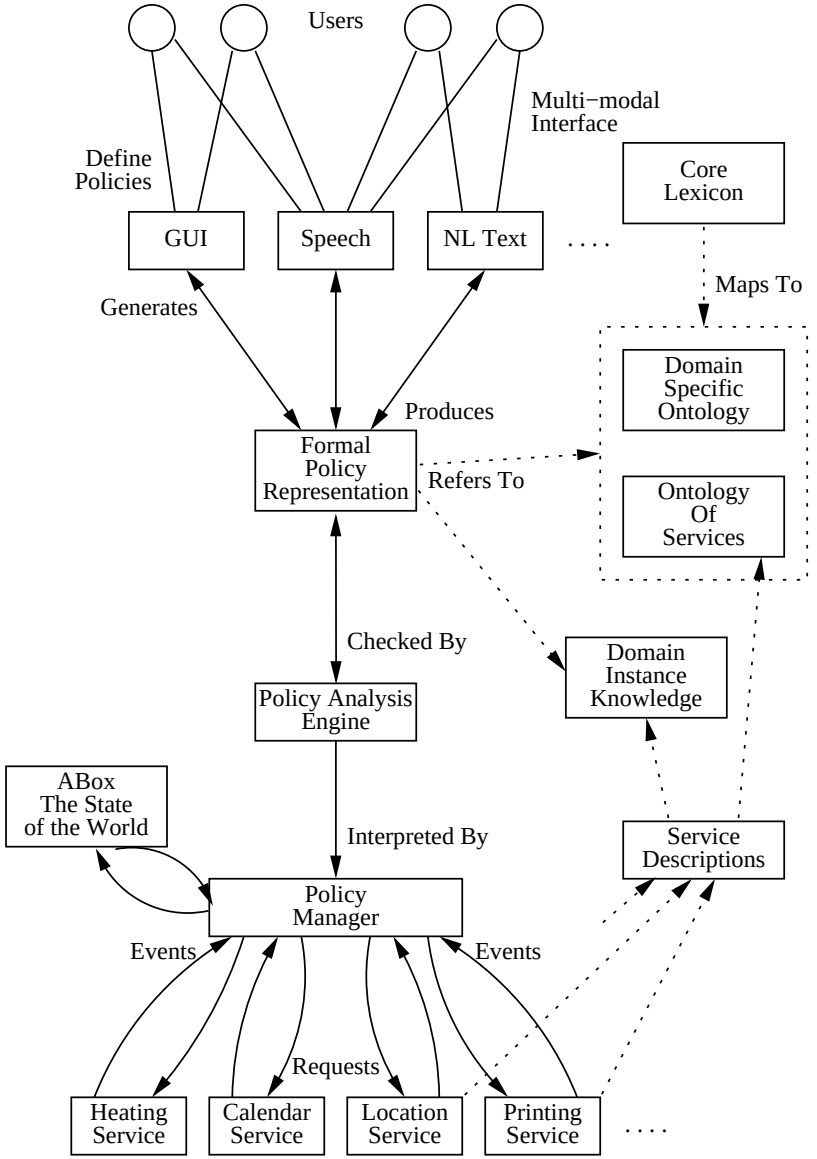
4 The Overall Architecture

Multi-disciplinary

- HCI
- NLP
- Network Services Ontologies

Avoiding

- AI, Goal-seeking etc.
- Automatic Service Composition



5 Policies in Description Logic

- NLP folk like a knowledge representation formalism
- Pervasive computing world is dynamic and open
- We need a way of explaining what is happening
- Description Logic seemed sensible (meets the above, and tools and theory available)
- KAOS and Rei similar use of DL

5.1 What is Description Logic?

- Describes set membership in concepts and relations eg

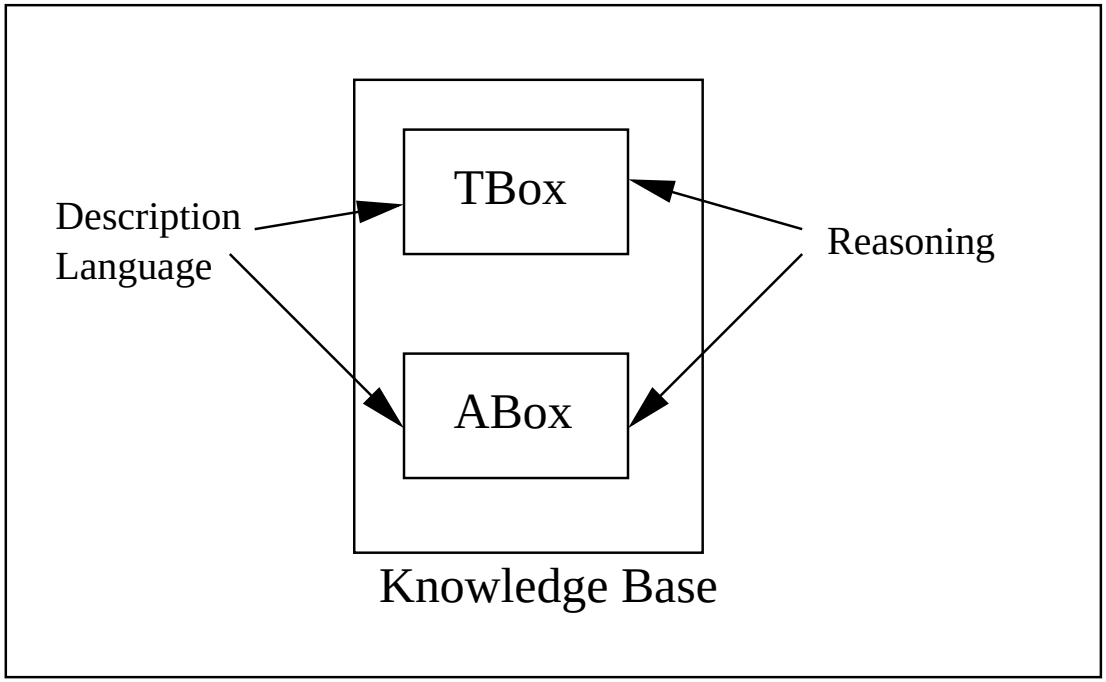
  $\text{Woman} \equiv \text{Person} \sqcap \text{Female}$

  $\text{Parent} \equiv \text{Person} \sqcap \exists\ \text{hasChild.Person}$

  $\text{Request} \sqsubseteq \text{Event}$

- Close relationship with predicate and other logics
- Good for expressing partial knowledge
- Good for incremental refinement
- Used heavily in knowledge engineering

5.2 TBoxes and ABoxes



TBox Terminological descriptions of concepts - the ontology

ABox The assertions abut the world - what facts are known

Reasoning Satisfiability, Subsumption, Equivalence, Disjointedness

5.3 Modes of inference in user policies

- Policies have
  - **pre-conditions** that express when the policy should be applied
  - **post-conditions** to be true when the pre-conditions hold
- *Over-write inference:* post-conditions overwrite contradictory existing facts
- *Default inference:* post-conditions are only asserted when they do not contradict pre-existing facts.

5.4 Policy Examples

Action Trigger When a request or event matches a pre-condition, generate a new request/event eg *When a print request goes to "ljx", email me.*

$x \in \text{Print} \sqcap\ target.name.\text{“}ljx\text{”} \Rightarrow$
$y \in \text{Email} \sqcap\ \text{target.name.“ianw”}$

Overwrite The request is changed eg *If a print request is sent to LJX and LJX is offline, then print to LJA instead.*

$x \in \text{Print} \sqcap\ target.(name.\text{‘LJX’} \sqcap\ status.\text{OFFLINE}) \Rightarrow$
$x \in target.name.\text{‘LJA’}$

Default If some fact is left unspecified, fill it in eg *The default printer for printing colour documents is COLJX.*

$x \in \text{Print} \sqcap\ patient.(\text{Document} \sqcap\ colourness.\text{COLOUR}) \overset{def}{\Rightarrow}$
$x \in target.name.\text{‘COLJX’}$

5.5 Policy Analysis

- When a new policy possibility is proposed
  - Use DL reasoning to check its satisfiable and realisable
  - Check for conflict using subsumption and disjointness over pre and post conditions
  - Ask user for further specification or use heuristics if there are problems
- When a policy is accepted, it is inserted into a policy partial order

5.6 Policy Execution

- Each notification and request is sequentially examined by policy engine wrt policy partial order
- State of the world is held in ABox, and tentative state in an inference stack
- If the pre-conditions of a policy hold when checked against ABox and stacks, then post-conditions are asserted onto the inference specific stack
- When policies are exhausted, notifications inserted into ABox, requests satisfied by meeting modified constraints of request

5.7 Current State

- Print policy corpus collected
- Natural Language implementation underway
- Java Racer implementation providing policy analysis and multiple modes of execution
- Middleware layer services (sms, email, etc) implemented (twice) over elvin and rmi (Jon Robinson).
- User study underway to determine how *smart* the policy engine should be

6 Conclusion

- Designing for users increases the level of abstraction in the middleware
- Description Logics are a promising avenue for policy management