Nathab: Policy management of pervasive computing services through natural language

Ian Wakeman       David Weir       Bill Keller
Julie Weeds       Tim Owen         Jon Rimmer
Jon Robinson      Thom Heslop

Dept of Informatics, University of Sussex

July 28, 2004
1 Overview

- Outline of project
- 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...
3.3 The Questions

- Can we use the current state of the art to convert natural language descriptions of policies into formal descriptions of software usable by machines?

- How should we collect a natural language corpus of future technologies?

- What are the requirements of service middleware of NL policies?

- What should the dialogue interface look like between the user and the machine clarifying policies?
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)...
- ...but no work in policy management used description logic that we could find.
5.1 What is Description Logic?

- Describes set membership in concepts and relations eg
  
  Woman ≡ Person ⊓ Female
  
  Parent ≡ Person ⊓ ∃ hasChild.Person
  
  Request ⊆ Event

- Close relationship with predicate and other logics

- Used heavily in knowledge engineering
5.2 TBoxes and ABoxes

**TBox** Terminological descriptions of concepts - the ontology

**ABox** The assertions about the world - what facts are known

**Reasoning** Satisfiability, Subsumption, Equivalence, Disjointedness
### 5.3 Print World - Domain-specific Mid-Level Ontology

<table>
<thead>
<tr>
<th>Class</th>
<th>Subclass</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device</td>
<td>⊆</td>
<td>Physical</td>
</tr>
<tr>
<td>ComputingObject</td>
<td>⊆</td>
<td>Abstract</td>
</tr>
<tr>
<td>User</td>
<td>⊆</td>
<td>Human</td>
</tr>
<tr>
<td></td>
<td>⊆</td>
<td>name:String</td>
</tr>
<tr>
<td>DataType</td>
<td>⊆</td>
<td>ComputingObject</td>
</tr>
<tr>
<td>String</td>
<td>⊆</td>
<td>DataType</td>
</tr>
<tr>
<td>File</td>
<td>⊆</td>
<td>ComputingObject</td>
</tr>
<tr>
<td></td>
<td>⊆</td>
<td>owner:User</td>
</tr>
<tr>
<td>Event</td>
<td>⊆</td>
<td>ComputingObject</td>
</tr>
<tr>
<td>Notification</td>
<td>⊆</td>
<td>Event</td>
</tr>
<tr>
<td>Request</td>
<td>⊆</td>
<td>Event</td>
</tr>
<tr>
<td></td>
<td>⊆</td>
<td>state:StateValue</td>
</tr>
<tr>
<td>StateValue</td>
<td>=</td>
<td>{STOPPED,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CANCELLED, COMPLETED}</td>
</tr>
</tbody>
</table>
### 5.4 Print World - Printing Service

<table>
<thead>
<tr>
<th>Printer</th>
<th>Device</th>
<th>DuplexVal =</th>
<th>{ SINGLE, DOUBLE }</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>\textit{name}.String</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>\textit{owner}.User</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>\textit{colour}.ColourVal</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>\textit{status}.StatusVal</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>\textit{duplicity}.DuplexVal</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Document</th>
<th>File</th>
<th>SecureVal =</th>
<th>{ OPEN, CONFIDENTIAL }</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>\textit{security}.SecureVal</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>\textit{colour}.ColourVal</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ColourVal</th>
<th>{ COLOR, MONO }</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>StatusVal</td>
<td>{ BUSY, ONLINE, OFFLINE }</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Print</th>
<th>Request</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>\textit{agent}.User</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>\textit{patient}.File</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>\textit{target}.Printer</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LJA</th>
<th>Printer</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>\textit{colour}.MONO</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>\textit{duplicity}.SINGLE</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
5.5 Policy Examples

Policies either

- Change the description of the request
- or generate new requests/events

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.

\[
\text{Print } \sqcap \text{target.name.} \text{“ljx”} \Rightarrow \text{new } y \in \text{Email } \sqcap \text{target.name.} \text{“ianw”}
\]
**Default**  When a request is incompletely specified, fill in relations eg when a print request has no target, send it to “ljx”.

\[
\text{Print } \sqcap \text{target.}\perp \\
\Rightarrow \text{target.name.} \text{“ljx”}
\]

**Alternate Targets**  If the world is in this state, rethink what to do with the request when the target of a print request is busy, send it to a non-busy printer.

\[
\text{Print } \sqcap \text{target.busyness.} \text{“busy”} \\
\Rightarrow \text{target.busyness.} \text{“empty”}
\]

**Block**  Disallow some requests based on conditions, eg when a print request is from “Mark, stop it.

\[
\text{Print } \sqcap \text{owner.} \text{“Mark”} \\
\Rightarrow \text{state.} \text{“stopped”}
\]
5.6 The PBox: Rewriting Requests

- Requests (in Description Logic)
- PBox
  - Rewrite Rules
- Knowledge Base
- Satisfiable?
- SBox
  - Service Rules
- Service Composition Middleware
5.7 (I can’t get no) Satisfaction

- When a policy is fired, we treat the rewrite initially as an additional constraint on the request
- However, this can make the request non-satisfiable eg
  
  \[
  \text{Print } \text{if target.name.‘lja’ } \land \text{ target.busyness.‘busy’} \Rightarrow \text{target.name.‘ljb’}
  \]

  results in the request having two target names.
- Therefore before entering a policy, we check the satisfiability of the join of the pre and post-condition, to infer which facts are retracted
5.8 Constructing a Partial Ordering over Policies

- The order in which policies are applied to requests is obviously important.

- Description Logic allows the creation of a partial order over policies defining which policies are most specific, and which conflict.

- For policies $P_1$ and $P_2$, $P_1 \leq P_2$ if:
  1. The pre-condition of $P_1$ subsumes the pre-condition of $P_2$, if $P_2$ fires, then $P_1$ should be tested.
  2. If the pre-conditions of $P_1$ and $P_2$ are disjoint, then only the ancestors of the rules which fire should be tested.
  3. If a precedence between policies is imposed (below), test one then the other.
5.9 Conflict Resolution

- Users may choose conflicting policies. Can we detect conflicts? (Current state of play)
  1. If the pre-conditions of $P_1$ and $P_2$ are do not subsume and are not disjoint, then the policies require precedence ordering and may conflict.
  2. If the post-conditions of $P_1$ and $P_2$ involve separate conditions, then they don’t conflict

- Resolve through
  - Querying the user (through the multi-modal interface) for disambiguation and precedence
  - Using heuristics to generate an ordering, eg security is more important than anything else
5.10 Problems in conflict resolution

• If a high precedence policy rewrites and increases the generality of the request, lower precedence policies may fire. Is this acceptable?
5.11 Automatic Conflict Detection: An Example

- The initial policy is to print documents on lja
  1. Print $\sqcap$ target.⊥ $\Rightarrow$ target.name."lja"

\[
\text{Print} ^\wedge \text{target._} \\
\Rightarrow \text{target.name."lja"}
\]
5.11.1 Disambiguation

- The next policy is to print colour documents on ljb
  2. $\text{Print } \cap \text{patient.colour.COLOR } \Rightarrow \text{target.name."ljb"}$
- However, this is neither subsumed/subsumes 1. nor is disjoint
- The user must disambiguate or define an arbitrary ordering.
- Disambiguation chosen
  2’. $\text{Print } \cap \text{target} \perp \cap \text{patient.colour.COLOR } \Rightarrow \text{target.name."ljb"}$
Print \( ^{\text{target.\_}} \rightarrow \text{target.name."lja"} \)

Print \( ^{\text{target.\_} ^{\text{patient.colour.COLOR}}} \rightarrow \text{target.name."ljb"} \)
5.11.2 Disjointedness

- The next policy is to print documents targetted for lja on ljb if lja is busy
  3. Print $\Box \text{target.name."lja"} \Box \text{target.status.BUSY} \Rightarrow \text{target.name."ljb"}$

- This is disjoint from the policies 1 and 2’ since there is a specified target.
Print ^ target._
  −> target.name."lja"

Print ^ target._ ^ patient.colour.COLOR
  −> target.name."ljb"

Print ^ target.name."lja" ^ target.status.BUSY
  −> target.name."ljb"
5.11.3 Arbitrary Precedence

- *Don’t print colour documents on lja*

  4. Print $\cap$ target.name."lja"$\cap$ patient.colour.COLOR $\Rightarrow$
     target.name.$\perp$

- This is neither disjoint nor subsumed/subsumes policy 3.

- User chooses to give 3. a higher precedence
Print ^ target._
→ target.name."lja"

Print ^ target._ ^ patient.colour.COLOR
→ target.name."ljb"

Print^target.name."lja"^patient.colour.COLOR
→ target.name._

Print^target.name."lja"^target.status.BUSY
→ target.name."ljb"
5.12 Executing Policies

- A request starts at the bottom of the constructed lattice and each policy which matches the pre-condition has the rewrites applied.

- When all the relevant policies have been executed, the necessary sets of possible targets, patients etc are inferred.

- The service rule is then fired if possible, using (arbitrary?) choices to fill the relevant slots.
5.13 (I still can’t get no) Satisfaction

- Policy rewrites may make requests unsatisfiable
- eg requiring the document to be printed in my office on a colour printer, when I have no colour printer in my office.
- After each policy rewrite, test the request for satisfiability against the ABox.
- If non-satisfiable, flag it to the user, since it may indicate a real-world problem in policies.
5.14 Current State

- Print policy corpus collected
- Natural Language implementation underway
- Thought designs and rapid prototyping to get to this point on policy implementation
- Middleware layer services (sms, email, etc) implemented (twice) over elvin and rmi (Jon Robinson).
6 Conclusion

• Description Logic is a promising mechanism for dealing with policy management of the pervasive computing environment

• Application to other policy management areas possible