Organizing Ad Hoc Agents for Human-Agent Service Matching

Wouter Pasman
W.Pasman@ewi.tudelft.nl
http://www.cg.its.tudelft.nl/~wouter/

Mobiquitous Conference
August 24-26, 2004, Boston, USA
Introduction

Increasing number of agents:
- Home automation: light, blinds, doors, fridge, stereo, tv, coffeemachine, ...
- Web services: Travel planning, weather forecast ...
- Applications: text editors, photo editors, ...
- Communication: email, phone, web browsers, ...
- E-Commerce: payment systems, shopping assistants
- Health care: pacemaker adjustment, monitoring..
- Mobile agents: snack shop, riding shop, car agents
How does user find an agent?

A few approaches exist

- Every agent has its own special physical interface
- Browser with list/menu of agents, keyword search
- Ontology-based approaches
- Central mediator with NL interface
1. Physical Interface per Agent

Devices at home: select = handle physical interface
+ Optimal for device and average user
- Expensive, needs extra space, inflexible
  -> Thus, suboptimal for individuals
- Many agents have no natural, nearby or human-size physical counterpart (heating, lights..)
  -> where is the interface?
- You have to know how to get what you want and where the physical device and its interface is
2. List of Agents

Browse list/menu of agents, keyword search

Gets unworkable when

- There are many agents
- Agents doing similar jobs but with different constraints, scope etc
- Agents are not at a fixed location
- When physical location of the agent matters
- Words in the list are not consistent (eg, Chinese, Dutch and English agents?)
- User cannot determine from the words in the list which agent(s) suit his needs
3. Ontology-based approach

Two different mechanisms exist:

(1) Service discovery service: enables searching for agents delivering a certain service type

(2) Service abstraction: a resolver forwards a request to an available agent.

```
SELECT grounding(service)
FROM Cantonese
WHERE booking_date='02.10.2003' AND booking_time='8:00 pm' AND price_range='medium'
```
Problems for the user:
He doesn’t know and even want to know about
- ontologies
- service names
- programming agent-search queries
- communication protocols

Technical problems in massive adhoc agent worlds:
- Straightforward implementation is not robust and efficient enough in ad-hoc networks
- Risk of network flooding
- Excessive caching and update work at each node
4. Central NLI Interpreter

Central natural language interface interpreting all user requests and orchestrates the agents
- Not ad-hoc: new agent may offer service not fitting into the knowledge of central interpreter

- Massive central knowledge/rule planning system needed, seems unfeasible at this moment

- Natural language interpretation is not robust when all vocabularies of all agents are combined

- Central translation ignores domain knowledge available in the agents (e.g., lights that are already on)
Our Proposal

Use **Natural Language** and **Context**

to support user in finding the appropriate agent

Context: usual tasks, location, gaze direction, task, plans, goals, history and agenda; detail knowledge.

(1) Distribute NL interpretation over the agents, to improve robustness of language understanding and to enable semantic interpretation

(2) Organize agents in ad-hoc network to enable context sensitive search
Architecture

Distributed NLI

Define NLI-ontology for agents understanding NL:
(1) Interpretation AttemptInterp(String user_request)
(2) Execute(Interpretation interp)

Interpretation contains
- value [0...1] for the understanding (syntactic fit)
- value [0...1] for the executability (semantic fit)

Agent has highly restricted scope so this should be quite straightforward for each agent
Distributed Context Knowledge

Every agent knows and can communicate about its context using a RelatedAgent ontology

Context for an agent:
- Area size it covers in the real world \([0...\infty]\)
- Other agents that have some relation with this agent
Three types of relations
1 TaskRelated (SimilarTask, StepOf, HasStep)
2 LocationRelated (In, Contains)
3 UserRelated (NearbyAgent, UserLooksAt, CommonTask,..)

Each agent can have many relations
Each relation also has a tightness [0...1]
Agents have to be asked individually about their relations
The relations **organize** the agents
ServiceMatcher agent

Matches a user request to the available agents
Uses our architecture to do the match
Negotiates with agents until good match is found or a problem arises
scope={PersonalAgent}
Repeat
{
    Ask interpretation(request) from agents in scope
    Wait long enough that all agents can reply
    Check #interpretations received
    {
        1 which is executable:
            activate that agent
        >1 executable:
            ask user which interpretation suits best
            many understood but no one executable
            fail: user probably asks something impossible
        0: if scope already extended 8 times
            then fail: limit of search space reached
            else scope=extend(scope)
    }
}
Service Matching Example

Example Use:
• user in kitchen,
• Just turned on the light
• now asks the Service Matcher
Prototype implementation

- FIPA compliant Java agents running on JADE/LEAP
- Ignore central Directory Facilitator
- Currently only coffeemachine doing full semantic interpr.
  Other agents match request with their vocabulary
- 522 Dutch agents spanning areas in Amsterdam
  running on 16 unix machines
- Search time = distance to matching agent
  * deadlinetime per ‘round’
DEMO on tuesday: 17:00 with smaller English version
Robustness

- Frame-based parsing for robust NL understanding
- Semantics and actual situation used for matching
- If agents drop out, we continue with remaining agents
  Hopefully an alternative agent is available
- Agent approach promotes robustness over centralistic system
  - agents can be moved away from troublesome hardware
  - no single point of failure
- Tight deadlines on requests: avoid waiting on replies
- Internal failures caught within the agent
  - agent survives; failure message to caller
  - agent may correct its feasibility estimates

BUT: caller maybe uncertain about status of request
Future Work

• WOz test of our system with real users
• Comparison of WOz study with performance of automatic servicematcher
• Other ways to support the user finding services