Supporting Dynamics in Service Descriptions – The Key to Automatic Service Usage

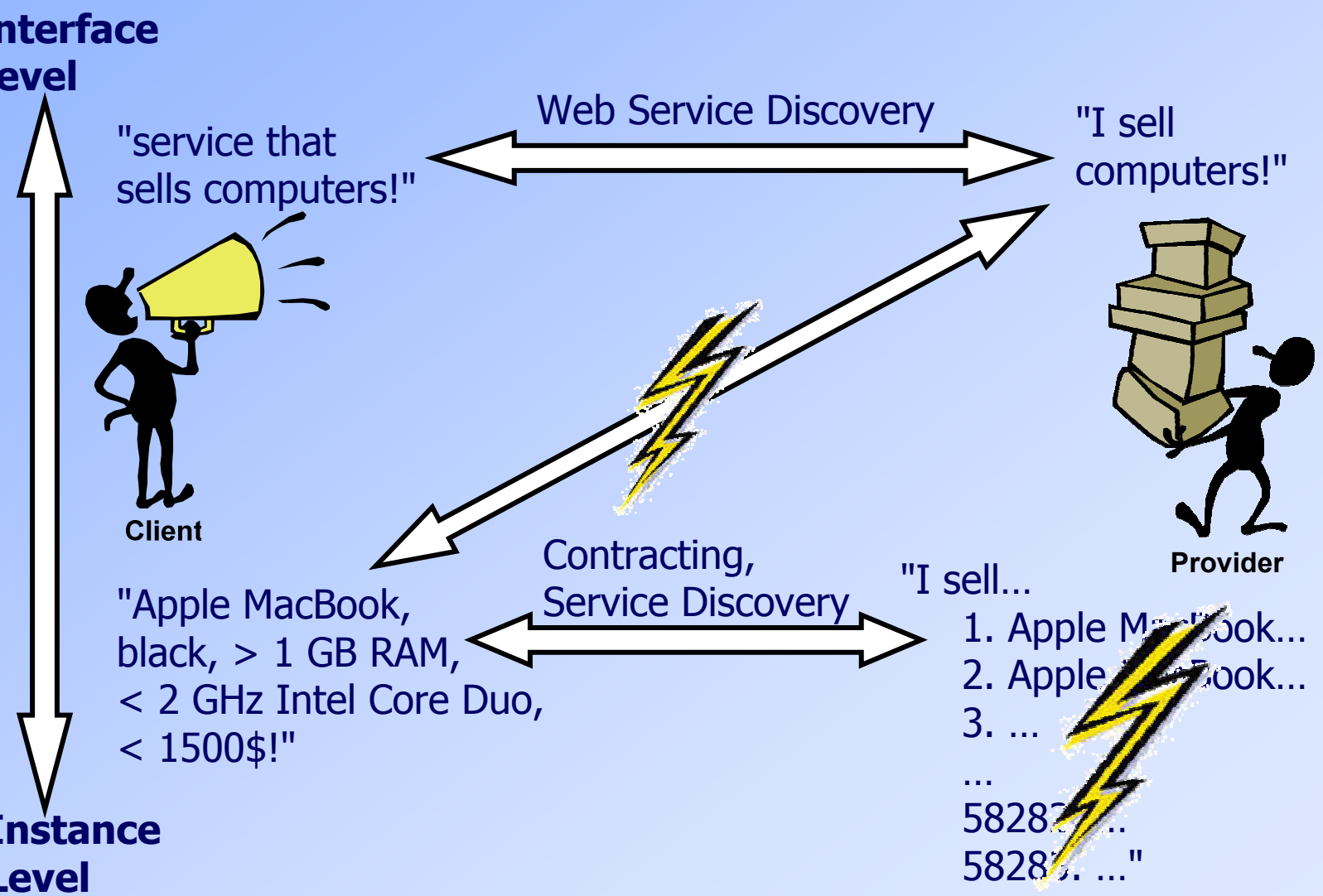
Ulrich Küster and Birgitta König-Ries
University Jena
Germany

ukuester|koenig@informatik.uni-jena.de

Introduction

- SOC / web services as promising computing paradigm
  - loosely coupled distributed systems
  - combine heterogeneous systems,
    ease switching of components
- Semantic Services
  - (semi-)automate time-consuming tasks (composition, selection, binding, …)
  - ➔ Leverage full potential of SOC
- Core Problem: Semantic Service Discovery

Motivation


Interface Level
"service that sells computers!"
Web Service Discovery
"I sell computers!"
Client
Provider
Contracting, Service Discovery
"Apple MacBook, black, > 1 GB RAM, < 2 GHz Intel Core Duo, < 1500$!"
"I sell...
1. Apple M...ook...
2. Apple ...Book...
3. ...
...
5828...
5828... ..."
Instance Level

Problems with exhaustive offer descriptions

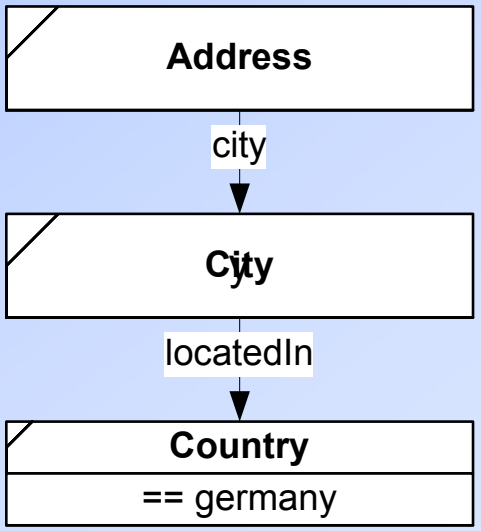
- amount of data
- dynamics, update problems

- data is money
- availability information constitutes bargaining power
  - seats for a flight
  - hotel rooms
  - …

➔ proposed solution: simply ask…
➔ dynamic offer descriptions
  ➔ gather necessary data at runtime
  ➔ follow human browse and choose paradigm
  ➔ illustration by means of DIANE Service Descriptions (DSD)

Characteristics of DSD - Sets

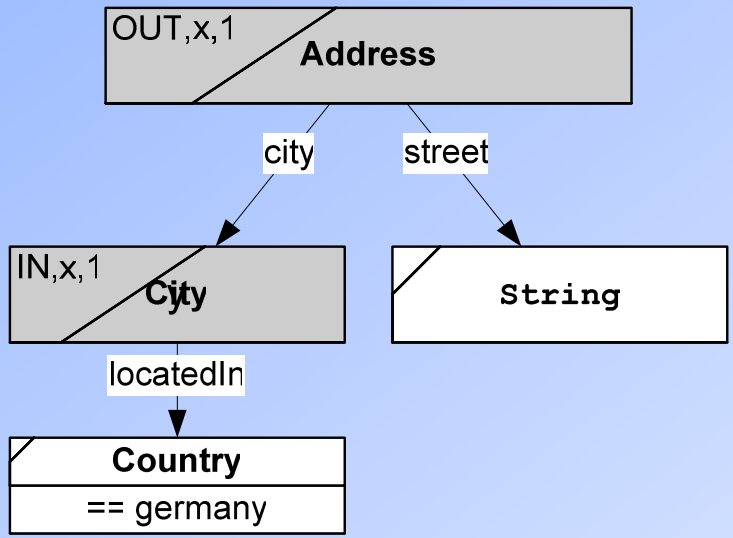
- **Set-based declarative descriptions**
  - Amazon sells millions of articles
  - Shippers provide transportation to a multitude of locations
  - ➔ Offer described as the set of possible effects
  - Requests envision perfect service, but accept different deviations
  - ➔ Request described as set of acceptable effects
  - ➔ Fuzziness to encode preference



➔ Standard semantics: One out of the described set of effects is requested / will be created

Characteristics of DSD - Variables

- **Need to configure offer** (which effect to provide)
- Variable = configurable set

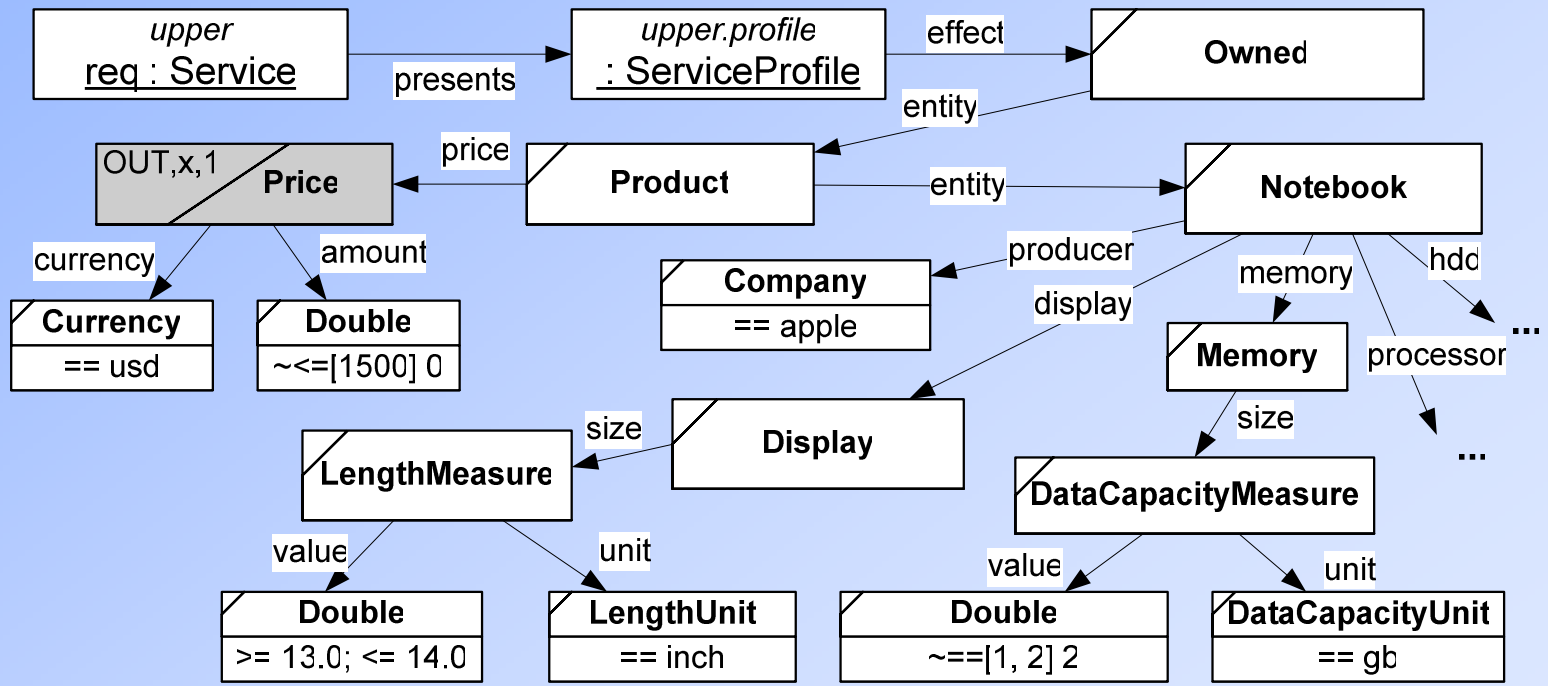


Input: German city

Output: Complete address in that city

Example service description

- 13" - 14" Apple MacBook, preferably 2 GB RAM, at most 1500 $, …

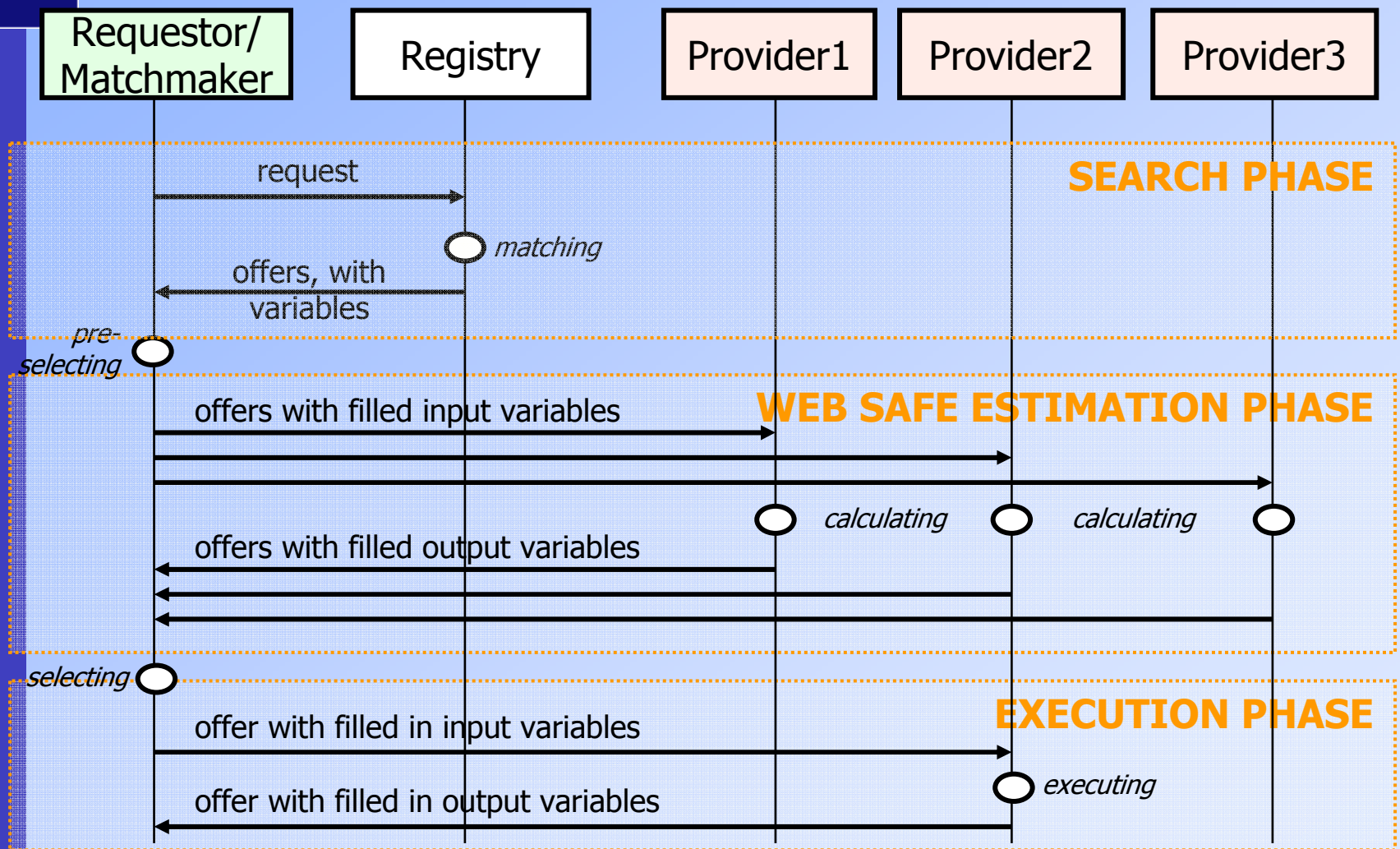


Matching DSD-Descriptions

- Given fuzzy request set ***r*** and configurable offer set ***o*** solve the following problem:
  a) Compute fuzzy containment value ***subset*** ∈ [0, 1] of ***o*** in ***r***
  (How well is the offer contained in the requested effects?)
  b) Where possible, **configure *o*** such as to **maximize subset**

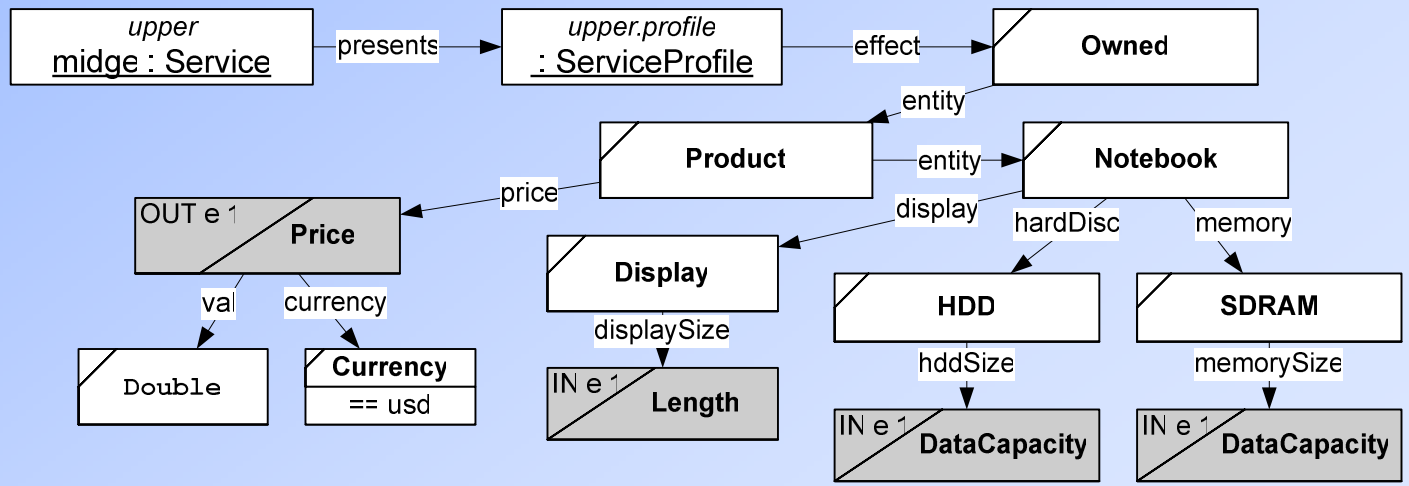
- Approach: Graph comparison
  - Implementation descends through description graphs
  - fills variables with optimal values
  - recursively computes subset for each element

Choreography: Staged Trading Process

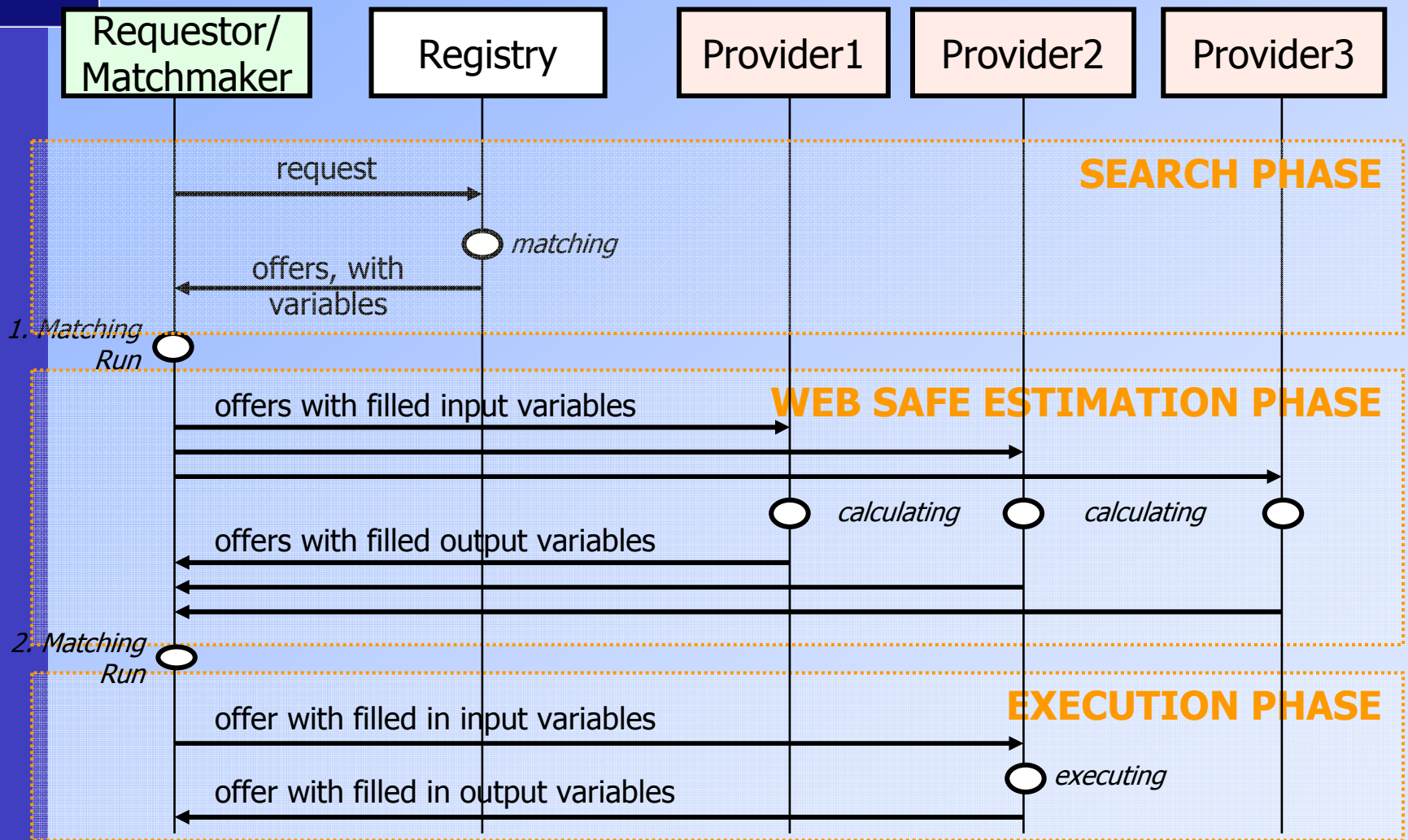


Estimation Phase

- Variables tagged `OUT,e,n` provide output during estimation phase
  - associated nth operation in grounding
  - associated `IN,e,n` variables provide input
- Example: Determine price of notebook based on configuration



Choreography: Staged Trading Process

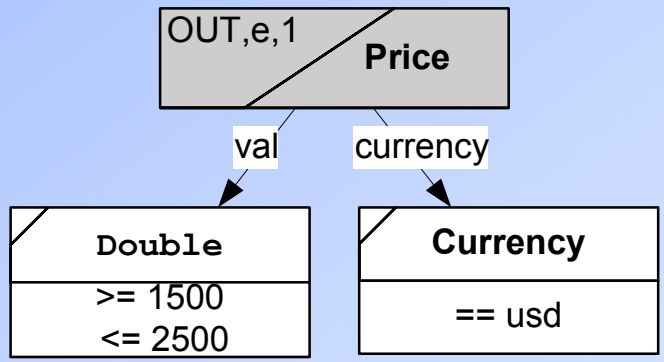


Integration into Matchmaking

- Phase 1: Matching on static descriptions
  - filter irrelevant offers
  - fill offer `IN` variables with values from request
  - collect information about usability of estimation operations
- Phase 2: Estimation phase
  - Execute only useful estimation operations (Communication is expensive!)
  - Update offer description with new information
- Phase 3: Final matching run
  - determine final input variable bindings
  - check for required outputs

Usability of Estimation Operations

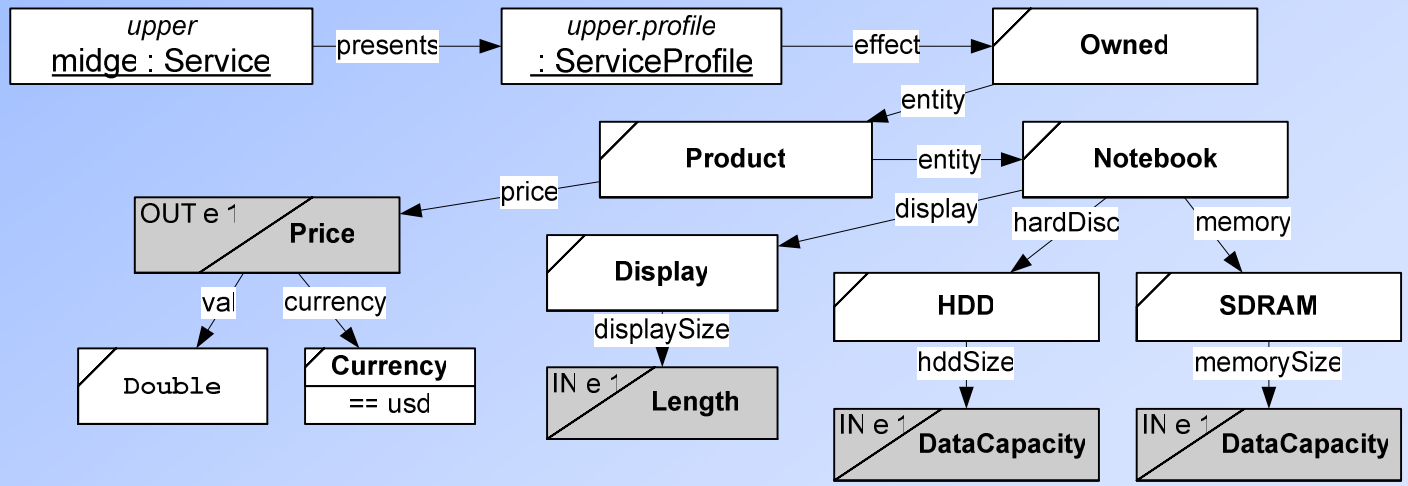
- Matcher encounters estimate `OUT` variable:
  1. Perfect match based on static description (no requested price or limit is greater than 2500$)
  2. Definite fail based on static description (requested price less than 1500$)
  3. Imperfect match or fail due to missing knowledge (requested price less than 2000$)



- Possible by
  - restricted expressivity of DSD (no arbitrary rules)
  - structured graph-matching approach

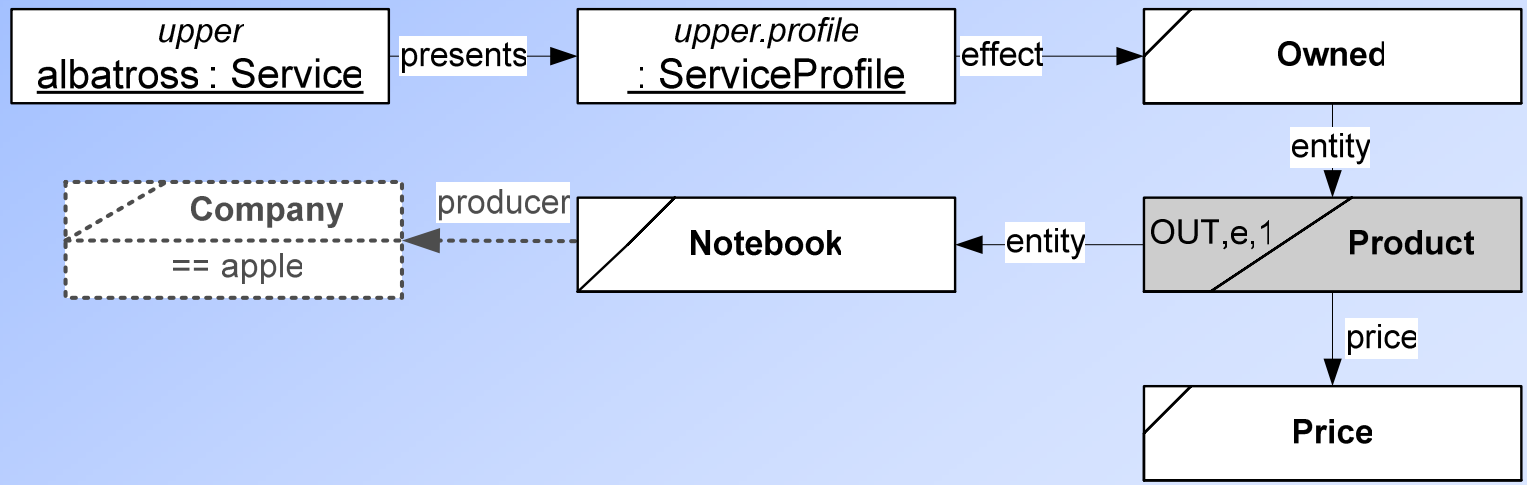
Shortcomings

- Matcher chooses values for variables
  - does not know typical dependencies
  - ➔ Potentially imperfect configuration



Example Albatross

- Albatross, big online shop, many articles
- human customers browse by category or search
- approach: One offer description per product type
- business decision: Never list more than 30 products

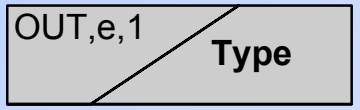


Extended variable concept: Binding Types

- Variables declare different supported bindings:

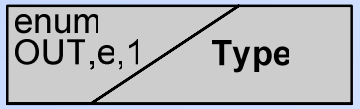
Simple Binding

➔ Variable filled with instance



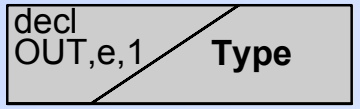
Enumerated Binding

➔ Variable filled with list of instances



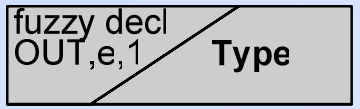
Declarative Binding

➔ Variable filled with crisp declarative set



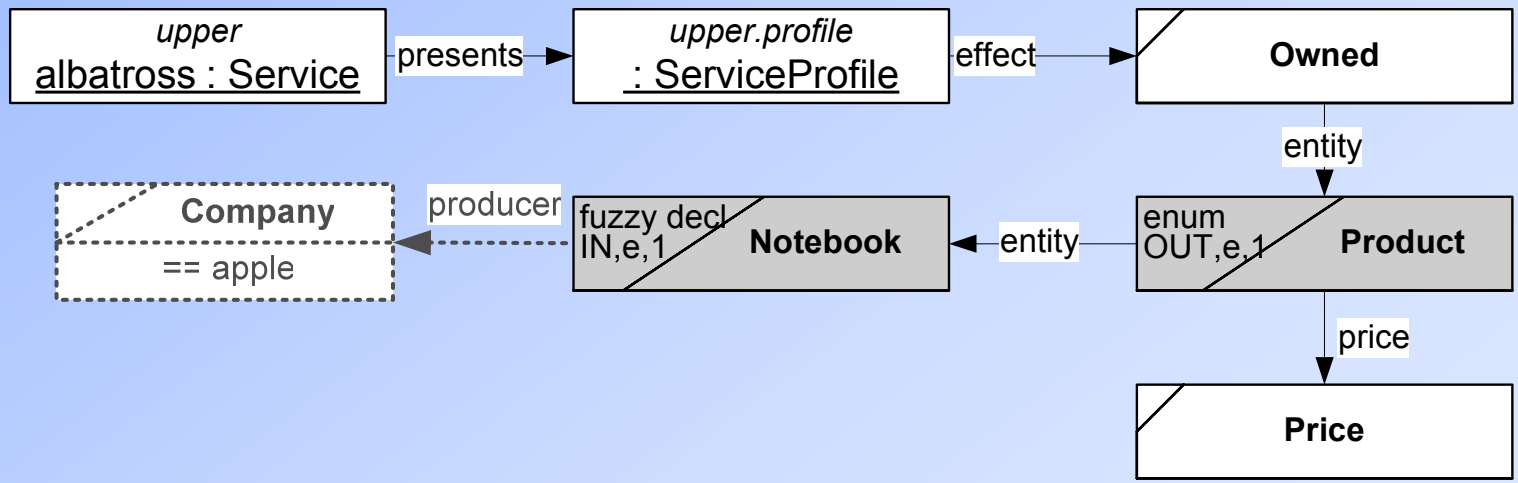
Fuzzy Declarative Binding

➔ Variable filled with fuzzy declarative set



Example Albatross revisited

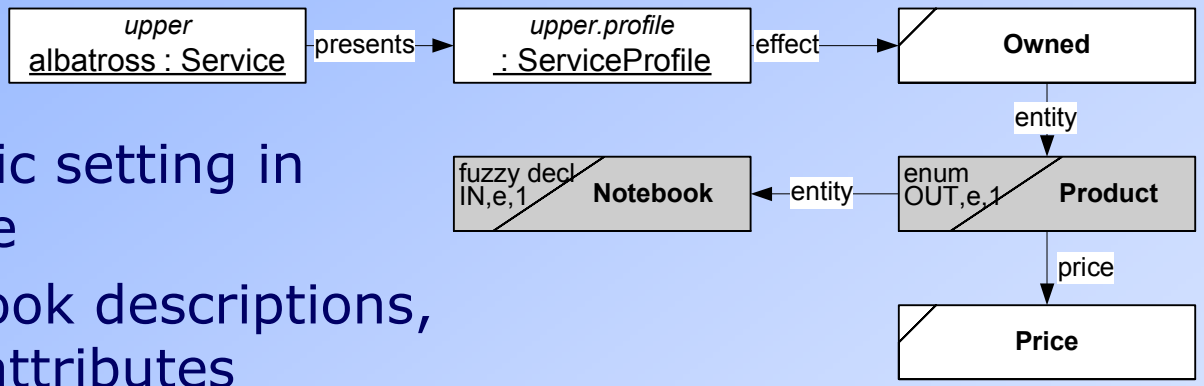
- Albatross, big online shop, many articles
- human customers browse by category or search
- business decision: Never list more than 30 products
- approach: One offer description per product type



External Evaluation

- First evaluation by SWS-Challenge
  - Initiative to develop certification process for SWS technology
  - Set of problem scenarios, evaluation at workshops
- Basic estimation operation used in first discovery scenario (dynamically inquire shipping price)
- New product purchasing scenario (retrieve product listing via declarative binding)
- Successfully evaluated at 4th SWS-Challenge workshop (June 2007)
  - Only solution able to determine usability of estimation
  - Evaluation results at www.sws-challenge.org
  - Solution including code online at http://sws-challenge.org/wiki/index.php/Solution_Jena

Internal Evaluation



- more realistic setting in terms of size
- 2500 notebook descriptions, > 100.000 attributes
- Cleaned and stored in relational database
- naïve implementation of Albatross, grounded to database
- fuzzy declarative binding to select 30 notebooks
- performance on Pentium 1,8 GHz machine
  - about 500 ms to query product database, create and load all 2500 instances
  - about 950 ms to determine precise match value wrt. requested notebook

Related Work

- Most practical approaches deal only with interface level discovery
- Conceptual work for contracting (but no details regarding its automation)

- Very few approaches regarding instance level
  - Two more SWS-Challenge solutions
    - WSMX-based (by DERI Galway)
    - Glue + WebML (by Politecnico Milano / CEFRIEL)
  - Web Service Matchmaking Engine (IBM, Facciorusso et al. 2003)
    - based on CORBA Trading Service
    - flat descriptions
  - no concept like binding types
  - no assessment of usability

Summary

- Approach for fully automated discovery and invocation on instance level
- Three-staged trading process
  - search phase (web service discovery)
  - estimation phase (service discovery / contracting)
  - execution phase (invocation)
- Automation of estimation phase by integration into matchmaking with assessment of usability
- Concept of Variable Binding Types for increased flexibility

Thank you for your attendance!

Questions?

Ulrich Küster
Ulrich.Kuester@uni-jena.de
http://hnsp.inf-bb.uni-jena.de/ukuester/
DIANE project (services in ad hoc networks)
http://hnsp.inf-bb.uni-jena.de/DIANE/