1st International Conference on Service Oriented Computing
Trento, December 15-18, 2003

Stepwise Refinable Service Descriptions: Adapting DAML-S to Staged Service Trading

Michael Klein, Birgitta König-Ries, Philipp Obreiter
Institute for Program Structures and Data Organization
Universität Karlsruhe, Germany

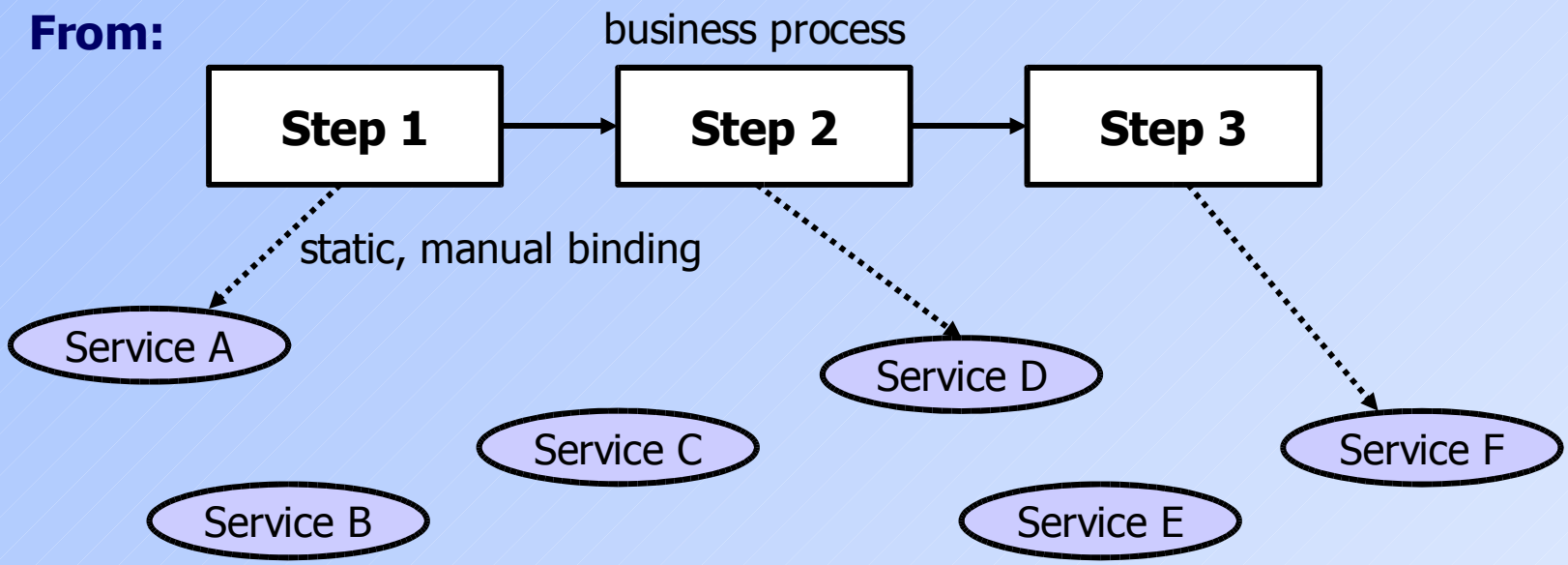
German Research Community
SPP 1140

DIANE Project

Motivation

- What is the main advantage/difference in service oriented computing?

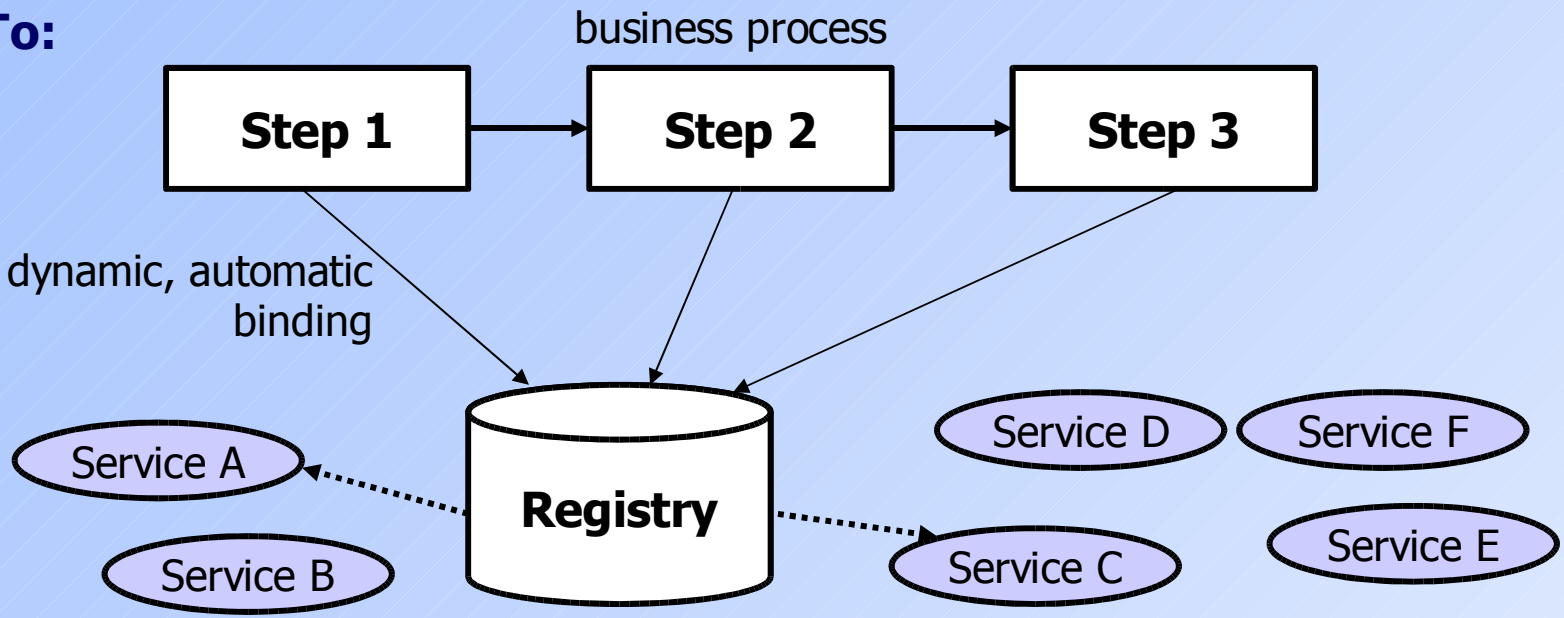
- **Agile Networks**
  → more efficient and robust business processes



Motivation

- What is the main advantage/difference in service oriented computing?
- **Agile Networks**
  → more efficient and robust business processes

To:



Automatic Service Binding

Automatic Service Binding needs a **computer-understandable service description**:

- 1) Abstract description of the **functionality** of the service
  - what does the service do?
- 2) Description of the **configuration process**
  - how to exchange information?

Example: Printing Service

Description of a printing service:

- **1) Functionality**
  - Transforms a document from state *LocallyAvailable* to state *Printed*
- **2) Configuration process**

  a) Requestor defines filesize of the document to print

  b) Provider calculates finishing time of the print process

  c) Requestor accepts the service and specifies the name of the file and the resolution of the printout

  d) After execution, Provider discloses where to find the printout

Service Description – State of the Art (1)

Message oriented service description

- Examples: IDLs (CORBA, DCOM, EJB), **WSDL**, ebXML, …

Printing service example:

Service Description – State of the Art (1)

Message oriented service description

- Examples: IDLs (CORBA, DCOM, EJB), **WSDL**, ebXML, …

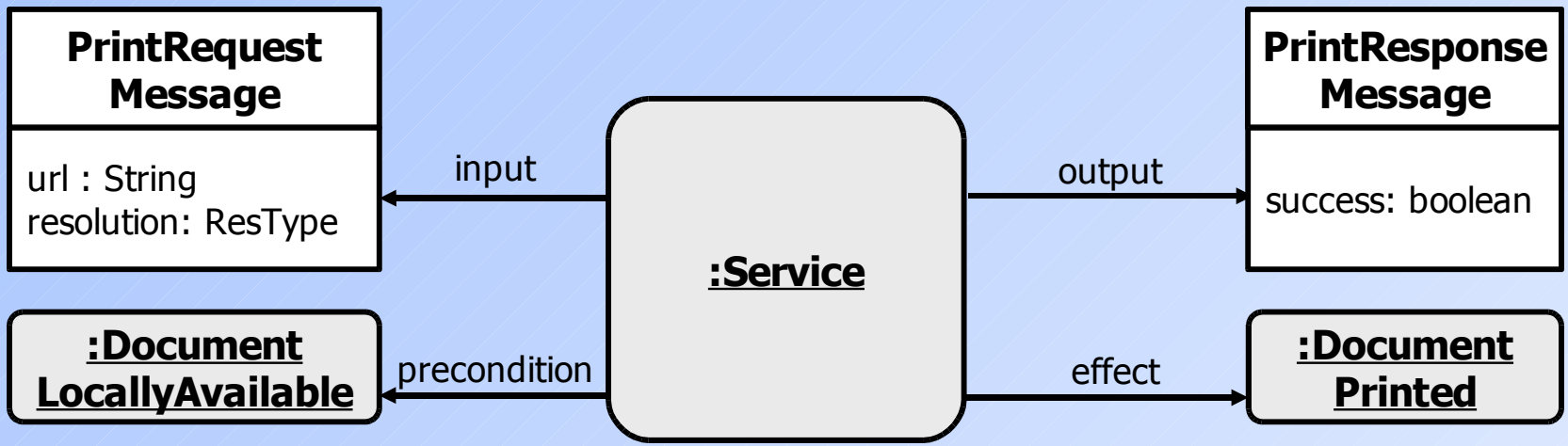
Problems

- Describe service by providing (simple) protocol only
  - 1) **Functionality**
    - Not given! What are the effects of the service?
    - → Guess semantics from message flow
  - 2) **Configuration process**
    - Just messages before and after service execution
    - no messages to decide if service is appropriate at the moment/in my case (printer queue?)

Service Description – State of the Art (2)

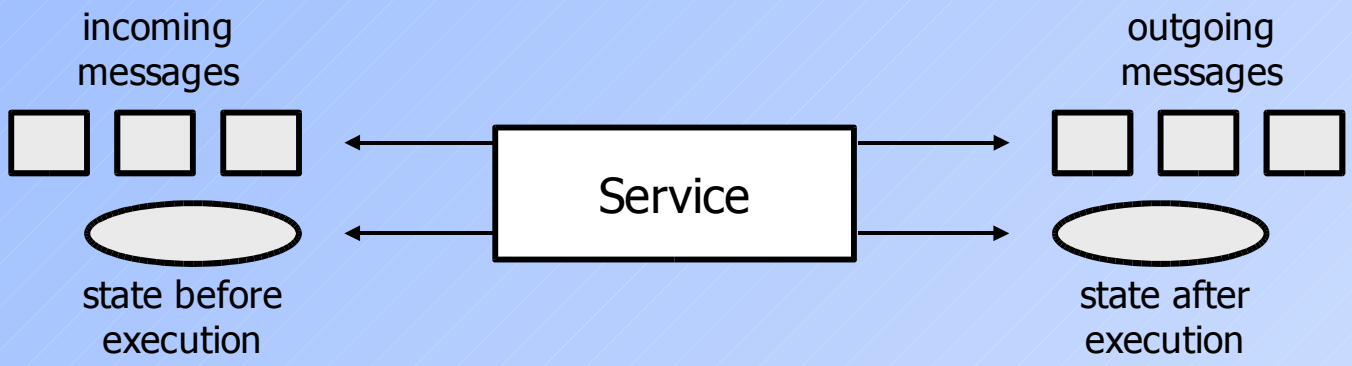
State/Message oriented service description

- Examples: DAML-S/OWL-S Profile, ICL, …



Service Description – State of the Art (2)

State/Message oriented service description



- Examples: DAML-S/OWL-S Profile, ICL, …

Problems

- Functionality and configuration process is described separately
  - 1) **Functionality**
    - Given, but how do messages influence functionality?
    - How are states related?
  - 2) **Configuration process**
    - Just messages before and after service execution
    - no messages to decide if service is appropriate at the moment or with my concrete parameters

Overview of the Problems

1) Functionality ⟷ 2) Configuration Process

Relation of states unclear

Influence of messages on states unclear

No well-defined **information exchange before execution** to decide if service is appropriate at the moment or with my concrete parameters

Overview of the Problems

1) Functionality ⟷ 2) Configuration Process

Relation of states unclear

↓

APPROACH 1:
Connect states via a shared object

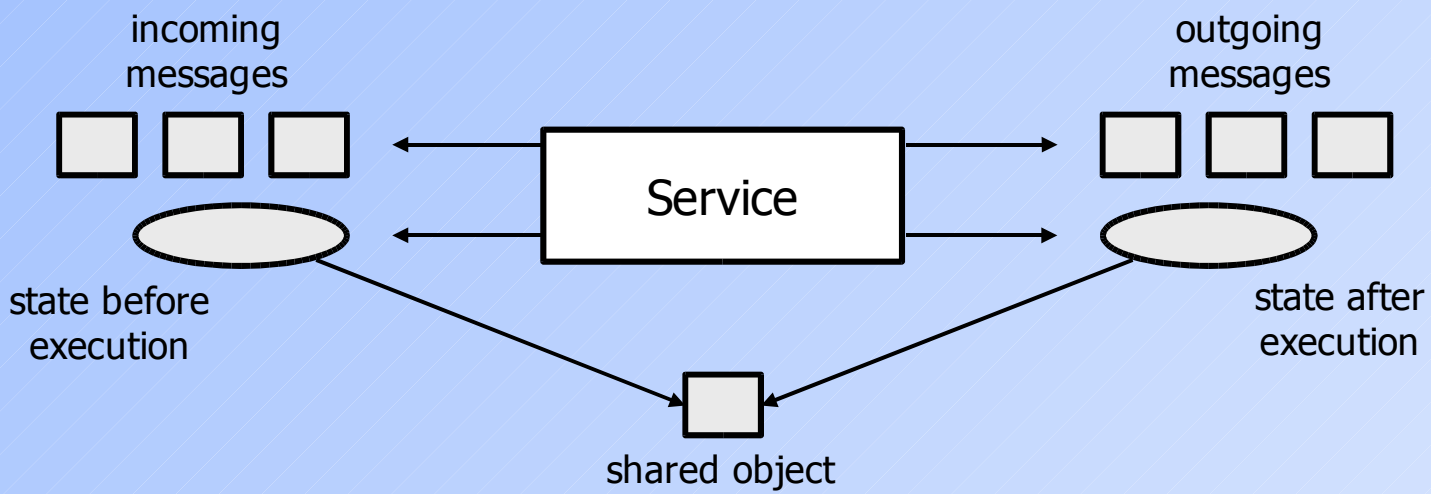
Influence of messages on states unclear

No well-defined **information exchange before execution** to decide if service is appropriate at the moment or with my concrete parameters

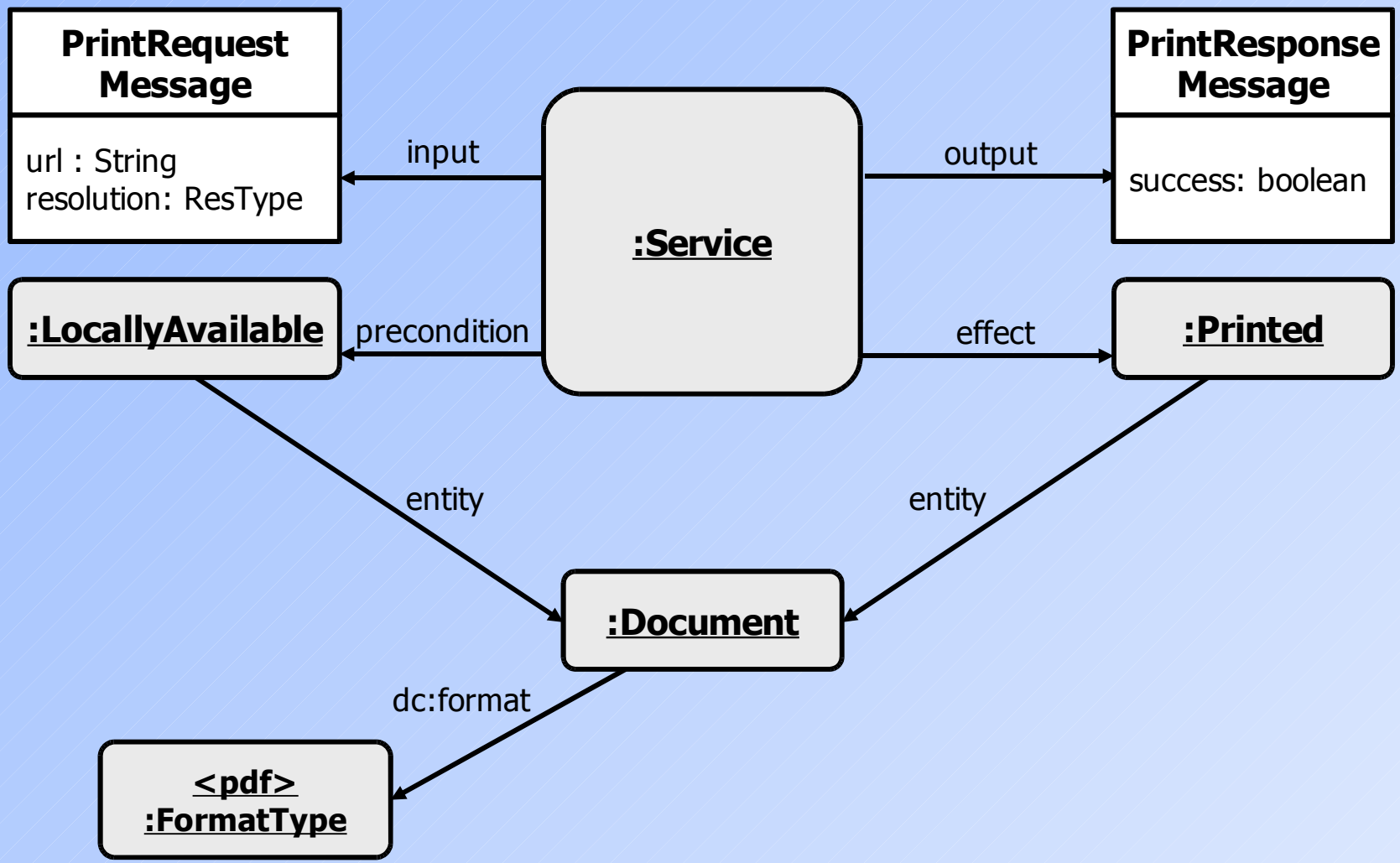
Approach 1

APPROACH 1

Connect states via a **shared object**.



Example with Approach 1



Advantages of our Approach

- **Interrelation** between precondition and effect is clearly visible
  → as they both point to a shared object

Overview of the Problems

1) Functionality ⟷ 2) Configuration Process

Relation of states unclear

Influence of messages on states unclear

↓

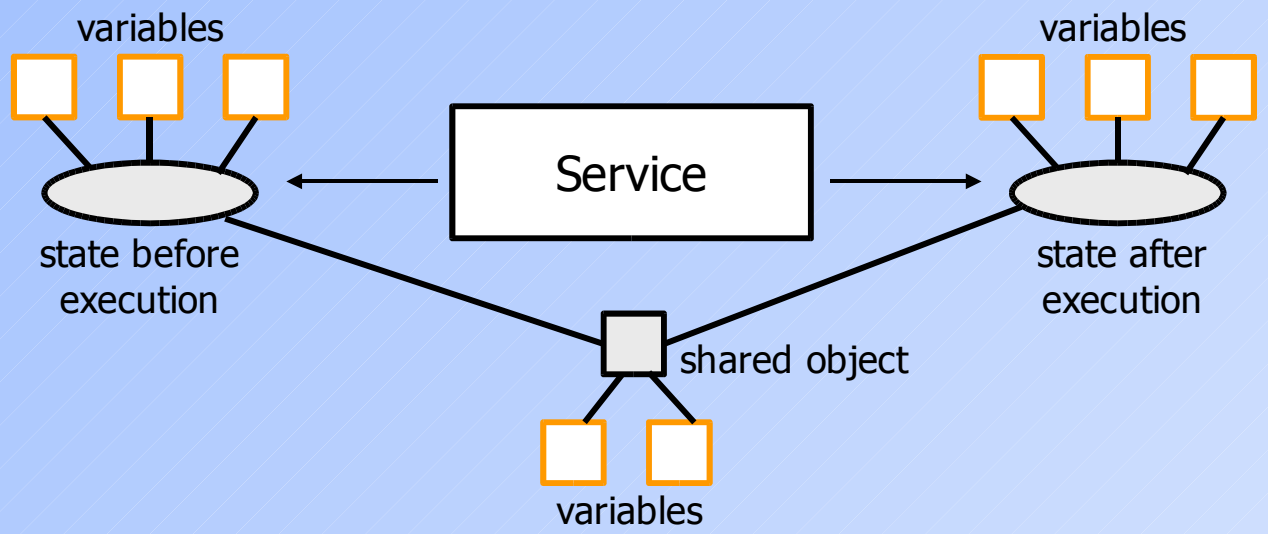
APPROACH 2: Pure state orientation with variables

No well-defined **information exchange before execution** to decide if service is appropriate at the moment or with my concrete parameters

Approach 2

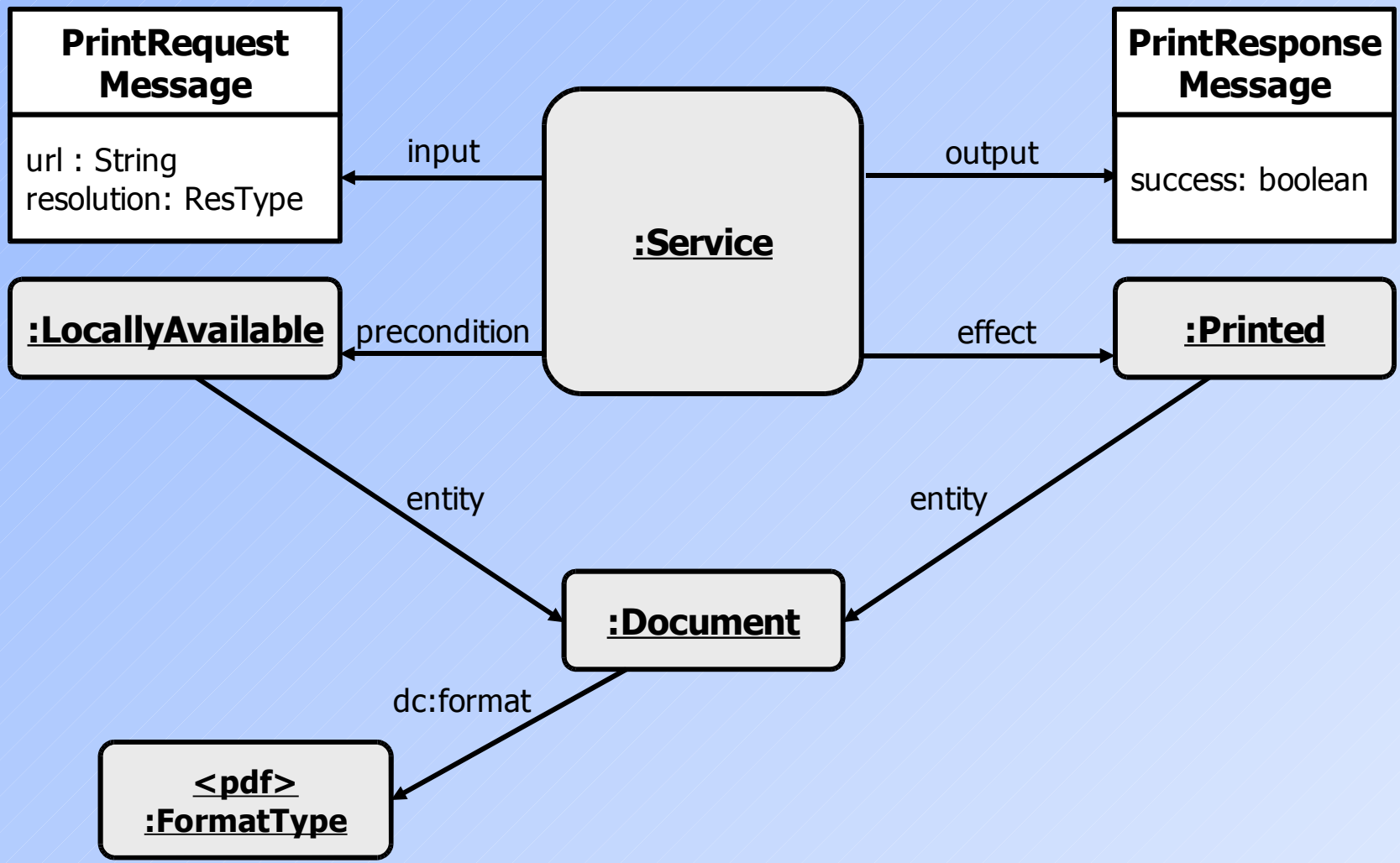
APPROACH 2

Transform into purely **state oriented** service description with **variables**.

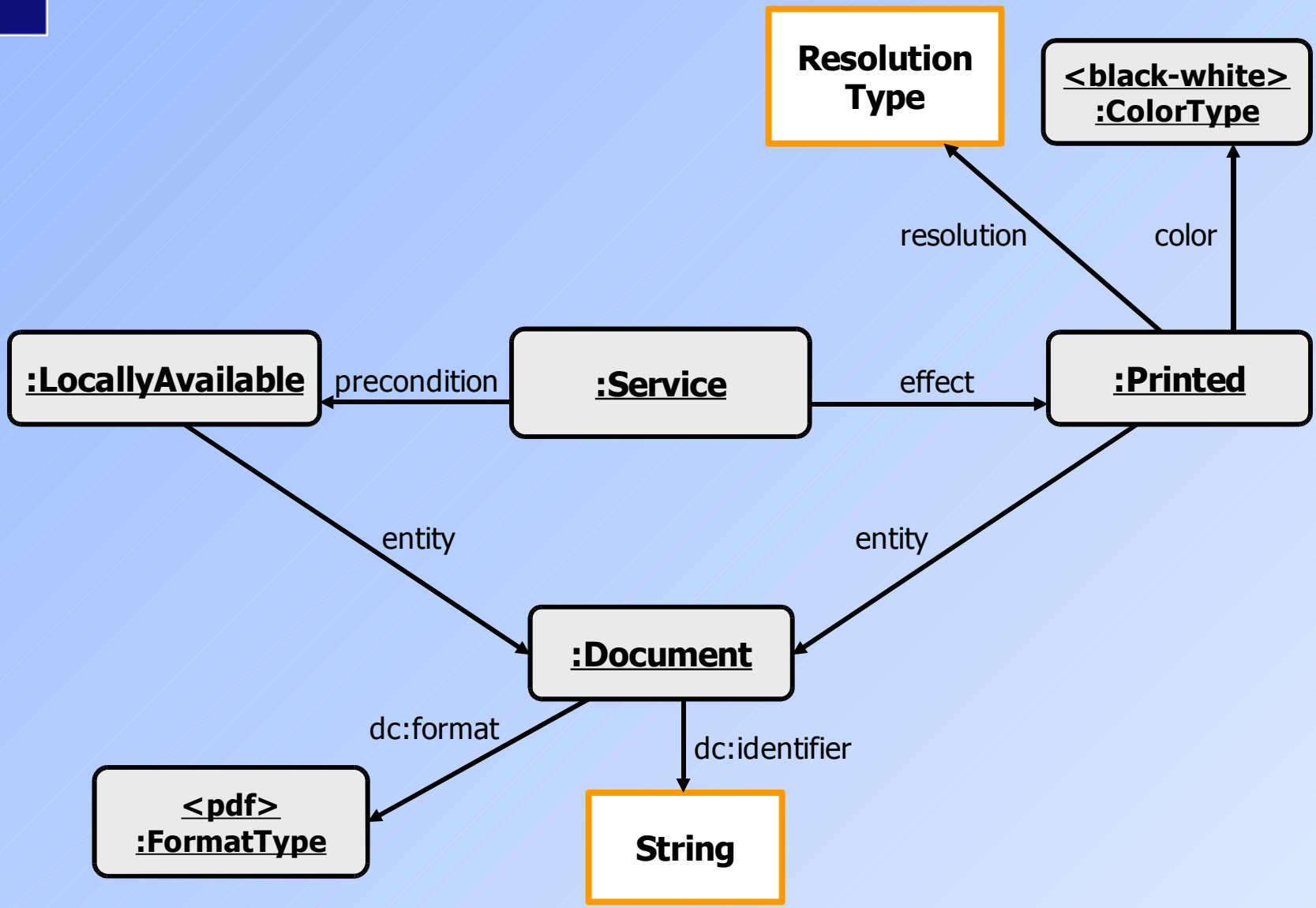


- Abandon use of separated messages
- Instead: Make states partly undefined by variables

Example with Approach 2



Example with Approach 2



Advantages of our Approach

- **Interrelation** between precondition and effect is clearly visible
  → as they both point to a shared object

- **Influence of parameters** on result is clearly visible
  → because of direct integration of variables into states
  → no separation between message and state

Overview of the Problems

1) Functionality ⟷ 2) Configuration Process

Relation of states unclear

Influence of messages on states unclear

No well-defined **information exchange before execution** to decide if service is appropriate at the moment or with my concrete parameters

↓

APPROACH 3: variable categories

Approach 3

APPROACH 3

Tag variables with **categories** to extend and clearly define configuration process.

Tag variables…

- **by whom** they have to be instantiated

IN **String** by requestor

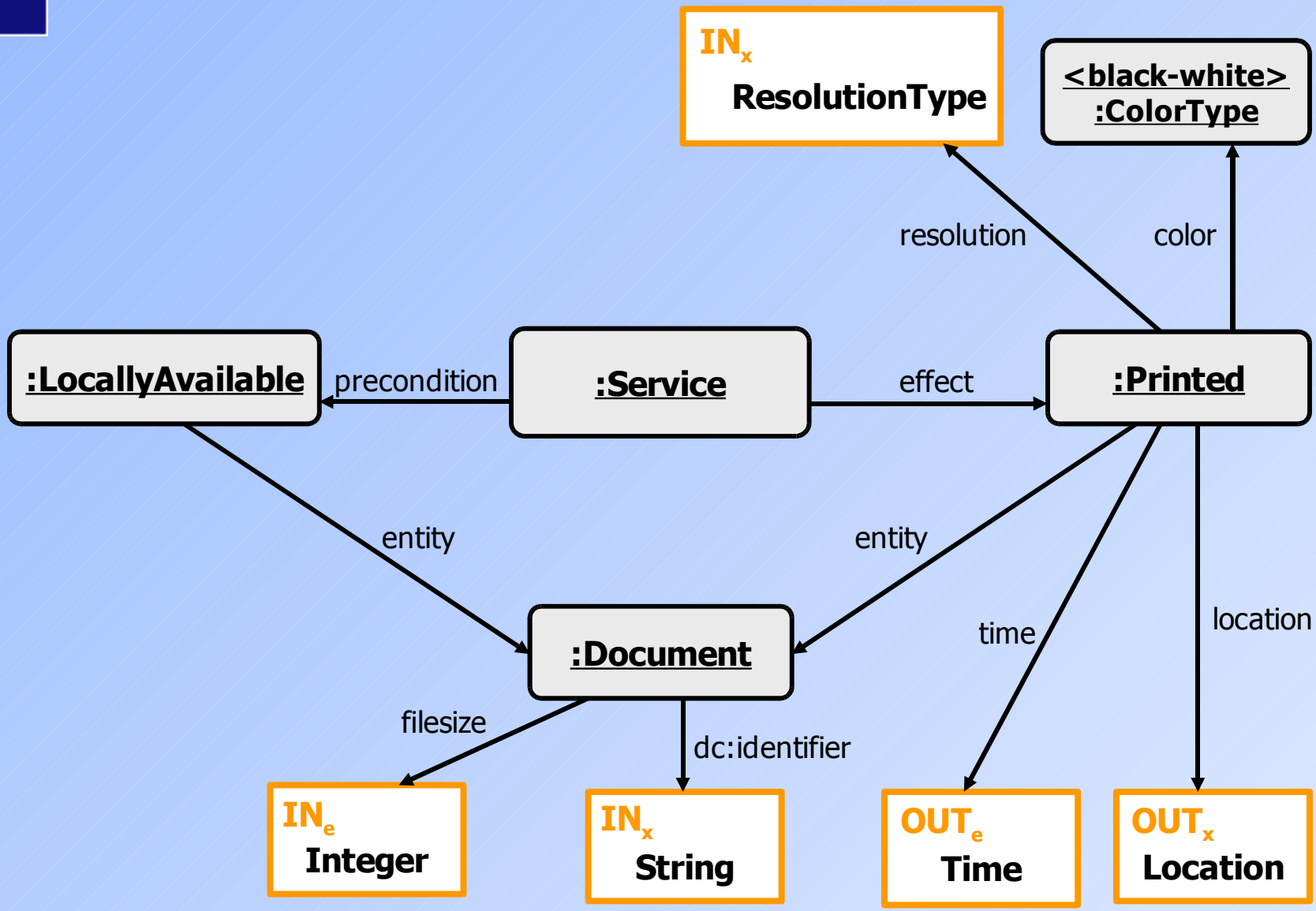
OUT **String** by provider

- **when** they have to be instantiated

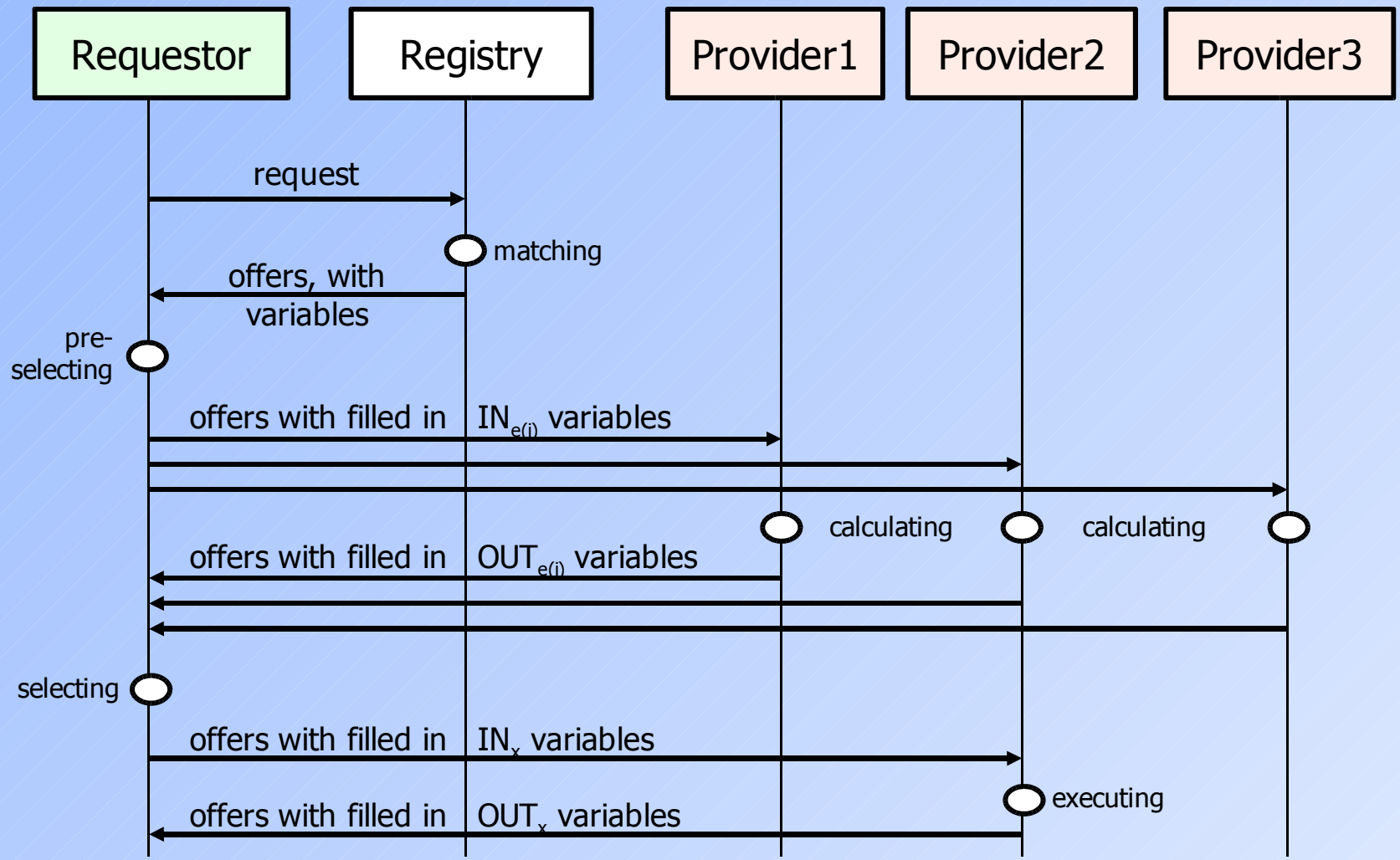
Before execution: fill in $\mathbf{IN}_{e(i)}$ to receive $\mathbf{OUT}_{e(i)}$
Fill in $\mathbf{IN}_{x}$ to execute service and receive $\mathbf{OUT}_{x}$

→ Configuration protocol is implicitly included in the description

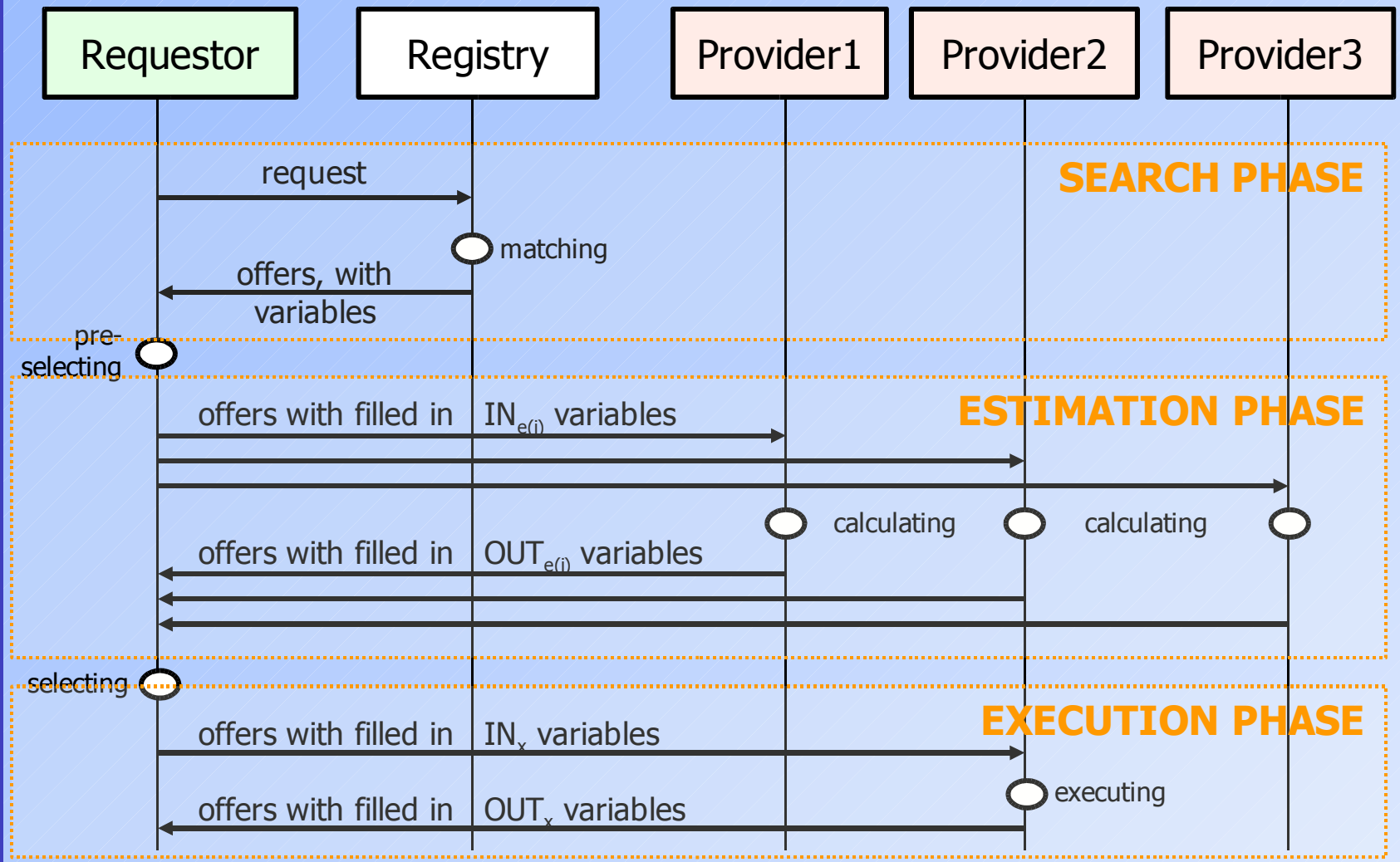
Example with Approach 3



Staged Trading Process



Staged Trading Process



Advantages of our Approach

- **Interrelation** between precondition and effect is clearly visible
  - → as they both point to a shared object
- **Influence of parameters** on result is clearly visible
  - → because of direct integration of variables into states
  - → no separation between message and state
- **Configuration process** is extended and clearly defined
  - → as variables have categories ($\texttt{IN}_e$, $\texttt{OUT}_e$, …)
  - → allows pre-execution configuration

→ **Configuration semantics** of the service **need not be guessed**, but is completely included in the description

Integration into DAML-S

- How can the concepts be **integrated into DAML-S**?
  - Problem: No variables in RDF/DAML
- **Requirements** for variables
  - variable = instance that has not been assigned a concrete value yet
  - name, type, category expressible
- Possible **technique**
  - Create rdf instance with rdf:ID, but don't assign value
  - express additional information by special properties

```
<rdf:Description rdf:ID="filesizevalue">
  <rdf:type>
    <rdf:Datatype rdf:about="xsd:integer"/>
  </rdf:type>
  <diane:varCat rdf:resource="#INe"/>
</rdf:Description>
```

Summary & Future Work

- Agile Networks need **computer-understandable service descriptions** to automatically bind services
- Existing languages don't have a clear configuration semantics
  - relation of states is unclear
  - influence of the messages on the states is unclear
  - no well defined information exchange before execution
- Three approaches
  - **shared objects**
  - no separated message/state description, but **pure state oriented description** with integrated **variables**
  - **variable categories** to extend and clearly define configuration process
- **Result: Configuration semantics is clearly visible in the service description**
- → Allows to automatically and dynamically find, configure and invoke appropriate services

- In the future:
  - Further improve querying possibilities
  - Implement matcher for those descriptions

...for your attention!

More information can be found on our website:
http://www.ipd.uni-karlsruhe.de/DIANE/en

APPENDIX

References

Michael Klein, Birgitta König-Ries
A Process and a Tool for Creating Service Descriptions based on DAML-S
4th VLDB Workshop on Technologies for E-Services (TES'03), Berlin

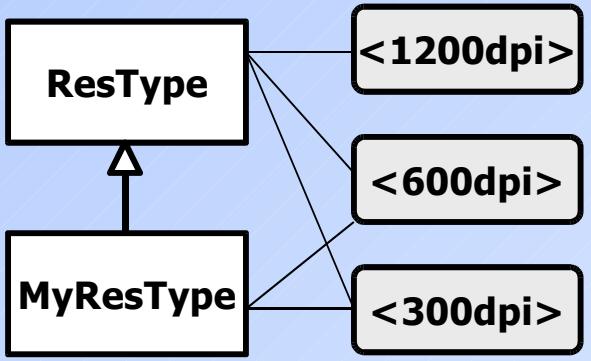
Michael Klein, Birgitta König-Ries, Philipp Obreiter
Stepwise Refinable Service Descriptions: Adapting DAML-S to Staged Service Trading
1st International Conference on Service Oriented Computing (ICSOC-03), Trento

Further Important Questions

- How can the **instantiation** possibilities of variables be **restricted**?
  - Problem: Not all legal values could be desirable
- How can the repository perform the **matching**?
  - Problem: Undefined variables in offer

Instantiation Restrictions

- How can the **instantiation** possibilities be **restricted**?
- Problem: Not all legal values could be desirable
  - offer does not allow/support all parameter values
- Two types:
  - Restrictions concerning a **single variable**
    - user-defined subtype
    - special restricting properties
  - Restrictions concerning **several variables**
    - non-orthogonal parameters → additional constraint list



?fs > 1000000 --> not ?r = <600dpi>

Matching with variables

- How can the repository perform the **matching**?
  - Problem: Undefined variables in offer
- **Idea**
  The repository returns all service offers that are **possibly matching** the request
- **Definition**
  Offer *o* and request *r* are **possibly matching** iff
  - all of *r*'s **effects** can be found in *o*
  - it exists a **valid variable binding** so that *o* and *r* are not contradictory