Policy-Based Service Selection

Helge Janicke and Monika Solanki
(heljanic@dmu.ac.uk / monika@doc.ic.ac.uk)

STRL, De Montfort University / DOC, Imperial College

YR-SOC 2007 11th June 2007 University of Leicester

Outline

1. Motivation

2. Service Architecture

3. Policy & Preferences

4. Example

5. Conclusion

Motivation

The *Why Policy-Based?* Question

- Quality of Service (QoS) is an ill-defined term.
- Service Level Agreements (SLA) define the QoS that a provider is expected to deliver.
- SLAs are legally binding, however it may or may not be feasible to take action.
- Some providers may guarantee less, but in fact provide "more" (or vice versa).
- Static SLA negotiation may not be useful if the system is characterised by: *uncertainty*, *failure* and frequent *re-configuration*.
- Instead: Best effort approach based on *dynamic* QoS attributes. QoS becomes *subjective*.
- How can we support this subjective QoS?

Motivation

The *Why Policy-Based?* Question

- Quality of Service (QoS) is an ill-defined term.
- Service Level Agreements (SLA) define the QoS that a provider is expected to deliver.
- SLAs are legally binding, however it may or may not be feasible to take action.
- Some providers may guarantee less, but in fact provide "more" (or vice versa).
- Static SLA negotiation may not be useful if the system is characterised by: *uncertainty*, *failure* and frequent *re-configuration*.
- Instead: Best effort approach based on *dynamic* QoS attributes. QoS becomes *subjective*.
- How can we support this subjective QoS?

Motivation

The *Why Policy-Based?* Question

- Quality of Service (QoS) is a defined term.
- Service Level Agreements (SLA) define the QoS that a provider is expected to deliver.
- SLAs are legally binding, however it may or may not be feasible to take action.
- Some providers may guarantee less, but in fact provide “more” (or vice versa).
- Static SLA negotiation may not be useful if the system is characterised by: *uncertainty*, *failure* and frequent *re-configuration*.
- Instead: Best effort approach based on *dynamic* QoS attributes. QoS becomes *subjective*.
- How can we support this subjective QoS?

Motivation

The *Why Policy-Based?* Question

- Quality of Service (QoS) is a defined term.
- Service Level Agreements (SLA) define the QoS that a provider is expected to deliver.
- SLAs are legally binding, however it may or may not be feasible to take action.
- Some providers may guarantee less, but in fact provide "more" (or vice versa).
- Static SLA negotiation may not be useful if the system is characterised by: *uncertainty*, *failure* and frequent *re-configuration*.
- Instead: Best effort approach based on *dynamic* QoS attributes. QoS becomes *subjective*.
- How can we support this subjective QoS?

Motivation

The *Why Policy-Based?* Question

- Quality of Service (QoS) is a defined term.
- Service Level Agreements (SLA) define the QoS that a provider is expected to deliver.
- SLAs are legally binding, however it may or may not be feasible to take action.
- Some providers may guarantee less, but in fact provide “more” (or vice versa).
- Static SLA negotiation may not be useful if the system is characterised by: *uncertainty*, *failure* and frequent *re-configuration*.
- Instead: Best effort approach based on *dynamic* QoS attributes. QoS becomes *subjective*.
- How can we support this subjective QoS?

Motivation

The *Why Policy-Based?* Question

- Quality of Service (QoS) is a defined term.
- Service Level Agreements (SLA) define the QoS that a provider is expected to deliver.
- SLAs are legally binding, however it may or may not be feasible to take action.
- Some providers may guarantee less, but in fact provide "more" (or vice versa).
- Static SLA negotiation may not be useful if the system is characterised by: *uncertainty*, *failure* and frequent *re-configuration*.
- Instead: Best effort approach based on *dynamic* QoS attributes. QoS becomes *subjective*.
- How can we support this subjective QoS?

Motivation

The *Why Policy-Based?* Question

- Quality of Service (QoS) is a defined term.
- Service Level Agreements (SLA) define the QoS that a provider is expected to deliver.
- SLAs are legally binding, however it may or may not be feasible to take action.
- Some providers may guarantee less, but in fact provide “more” (or vice versa).
- Static SLA negotiation may not be useful if the system is characterised by: *uncertainty*, *failure* and frequent *re-configuration*.
- Instead: Best effort approach based on *dynamic* QoS attributes. QoS becomes *subjective*.
- How can we support this subjective QoS?

Motivation

The *Why Policy-Based?* Question

- Quality of Service (QoS) is a subjectively defined term.
- Service Level Agreements (SLA) define the QoS that a provider is expected to deliver.
- SLAs are legally binding, however it may or may not be feasible to take action.
- Some providers may guarantee less, but in fact provide "more" (or vice versa).
- Static SLA negotiation may not be useful if the system is characterised by: *uncertainty*, *failure* and frequent *re-configuration*.
- Instead: Best effort approach based on *dynamic* QoS attributes. QoS becomes *subjective*.
- How can we support this subjective QoS?

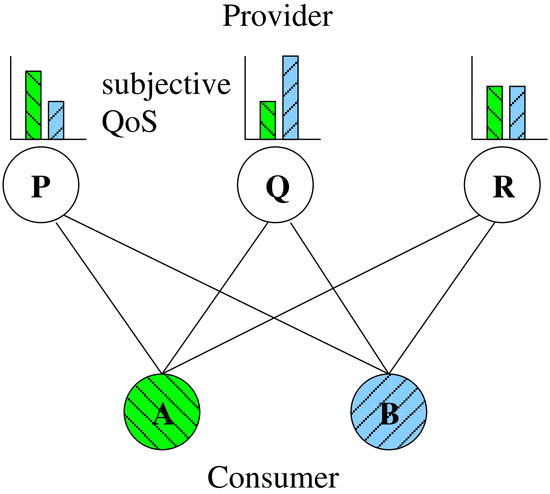
Motivation

The *Why Policy-Based?* Question

- Quality of Service (QoS) is a subjectively defined term.
- Service Level Agreements (SLA) define the QoS that a provider is expected to deliver.
- SLAs are legally binding, however it may or may not be feasible to take action.
- Some providers may guarantee less, but in fact provide "more" (or vice versa).
- Static SLA negotiation may not be useful if the system is characterised by: *uncertainty*, *failure* and frequent *re-configuration*.
- Instead: Best effort approach based on *dynamic* QoS attributes. QoS becomes *subjective*.
- How can we support this subjective QoS?

Subjective QoS Assessment

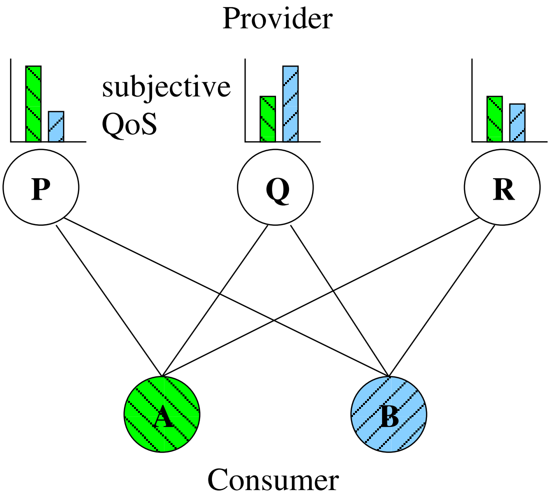
Subjective QoS is based on experiences.



Time = 0

Subjective QoS Assessment

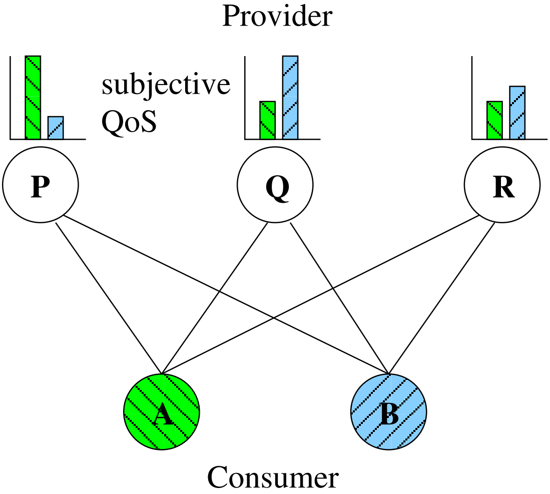
Subjective QoS is based on experiences.



Time = 1

Subjective QoS Assessment

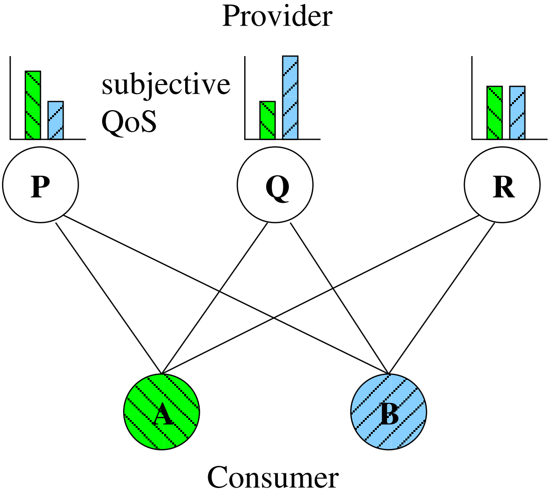
Subjective QoS is based on experiences.



Time = 2

Subjective QoS Assessment

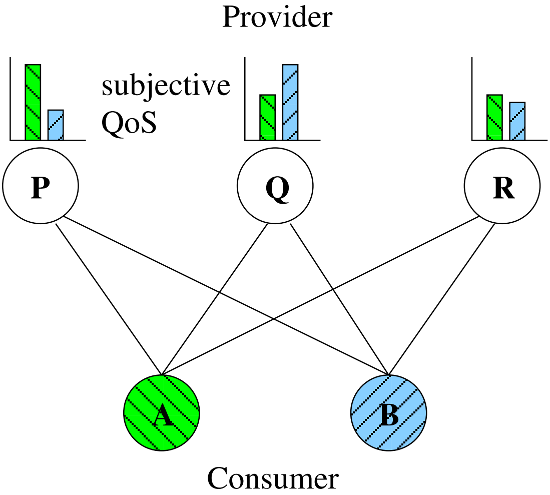
Subjective QoS is based on experiences.



Time = 3

Subjective QoS Assessment

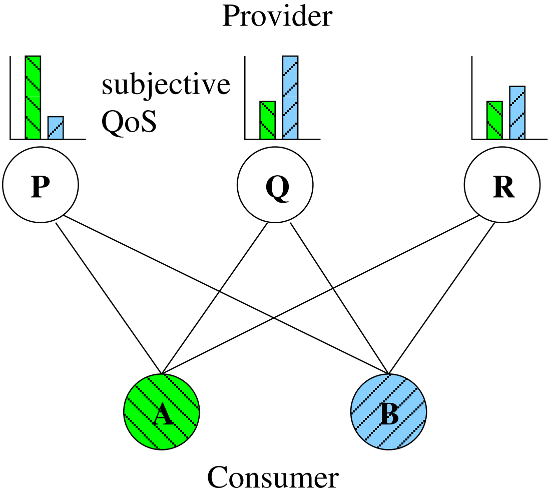
Subjective QoS is based on experiences.



Time = 4

Subjective QoS Assessment

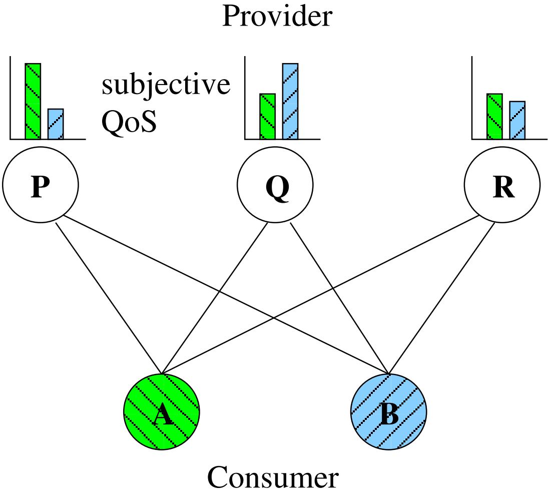
Subjective QoS is based on experiences.



Time = 5

Subjective QoS Assessment

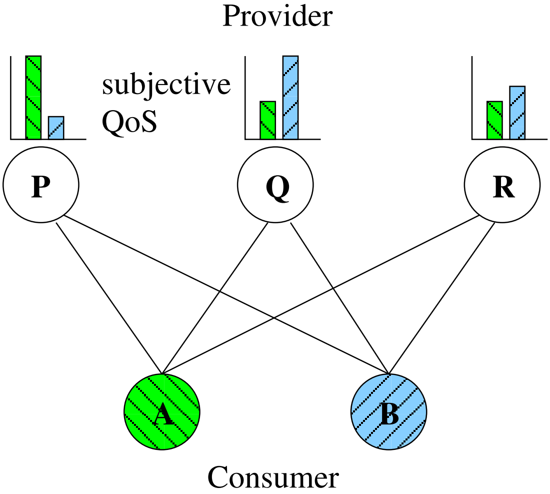
Subjective QoS is based on experiences.



Time = 6

Subjective QoS Assessment

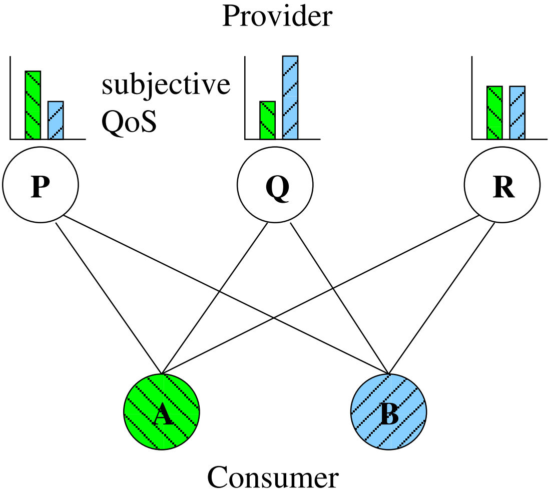
Subjective QoS is based on experiences.



Time = 7

Subjective QoS Assessment

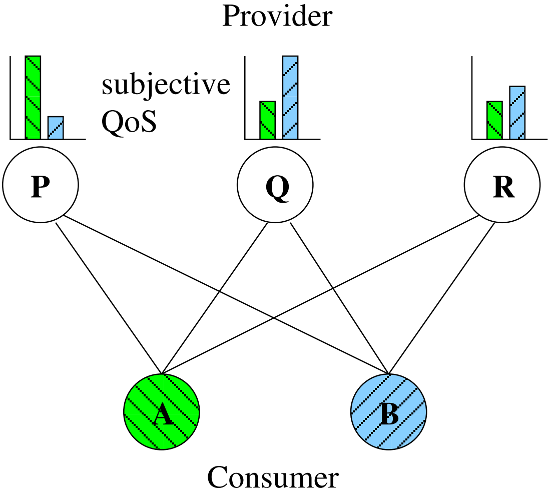
Subjective QoS is based on experiences.



Time = 8

Subjective QoS Assessment

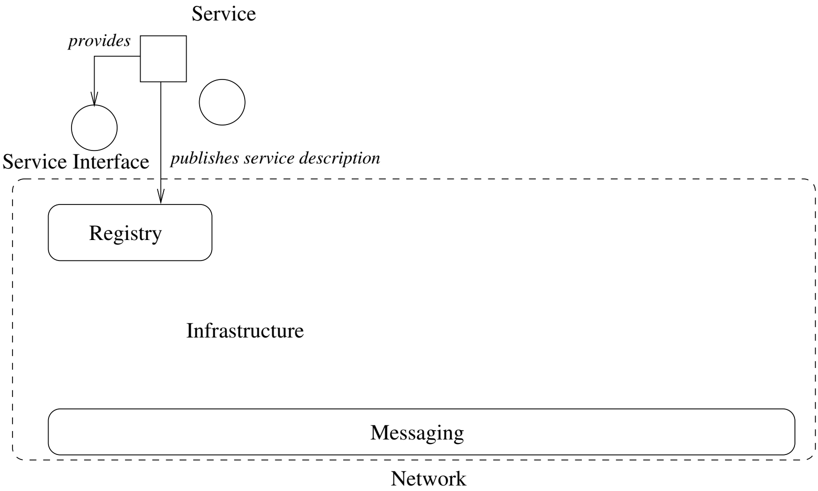
Subjective QoS is based on experiences.



Time = 9

Service Architecture

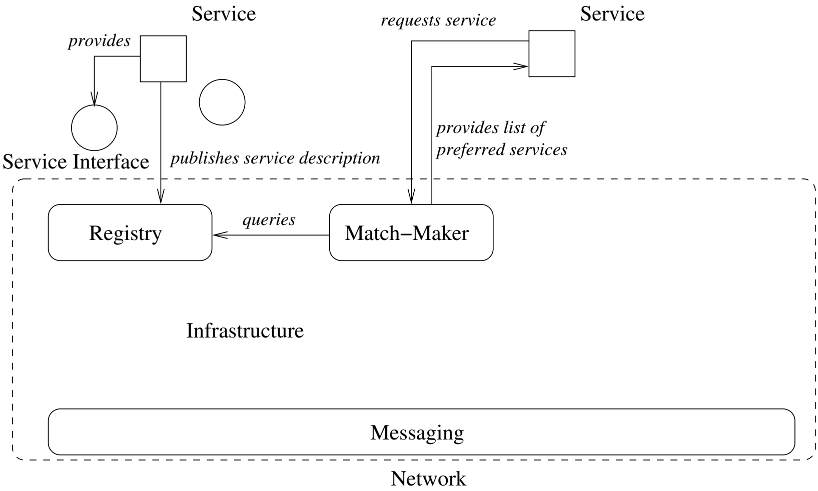
What components are required to base Service Selection on Dynamic QoS?



STRL

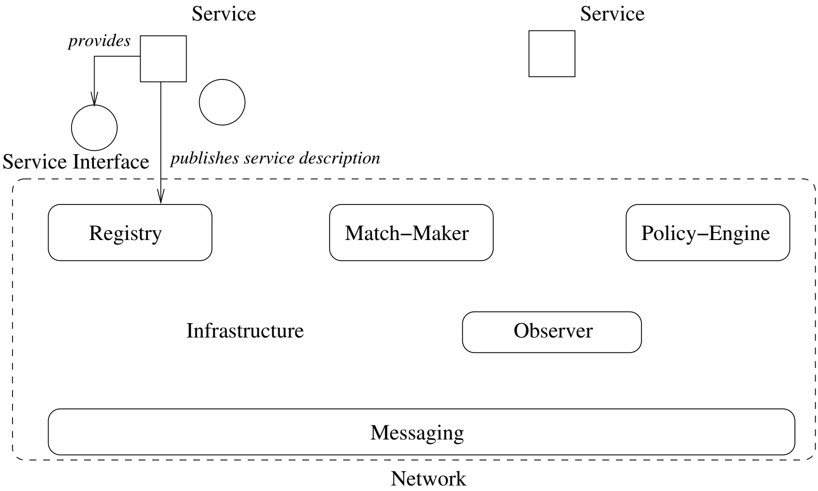
Service Architecture

What components are required to base Service Selection on Dynamic QoS?



Service Architecture

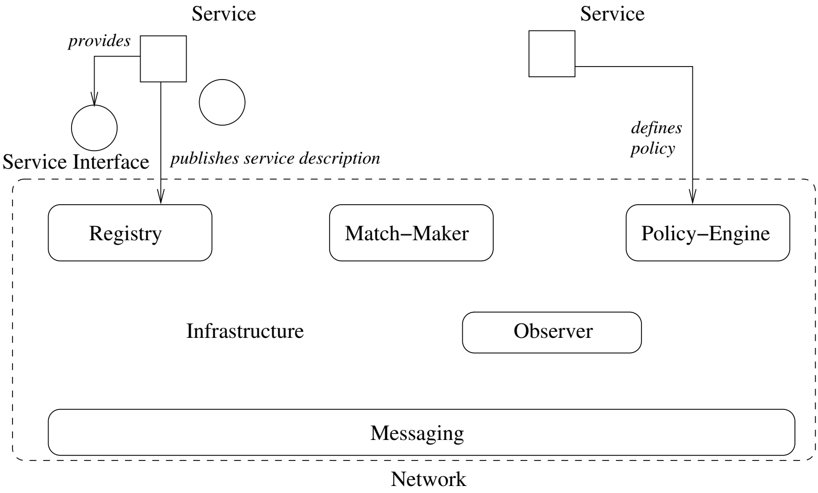
What components are required to base Service Selection on Dynamic QoS?



STRL

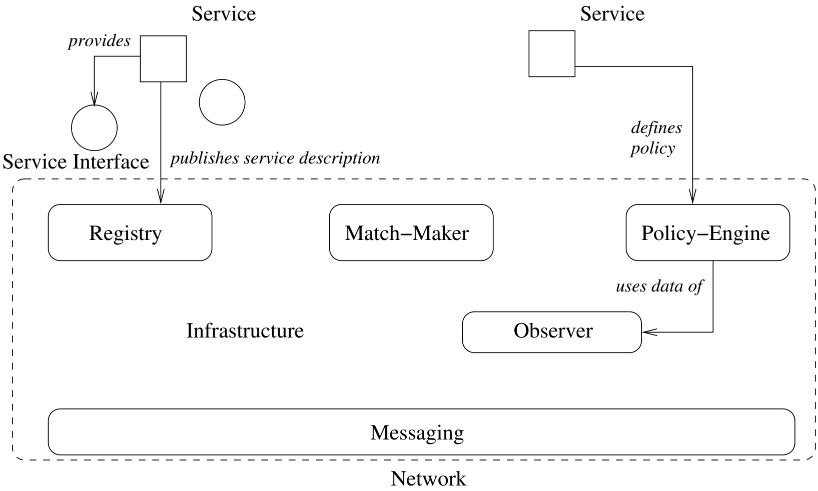
Service Architecture

What components are required to base Service Selection on Dynamic QoS?



Service Architecture

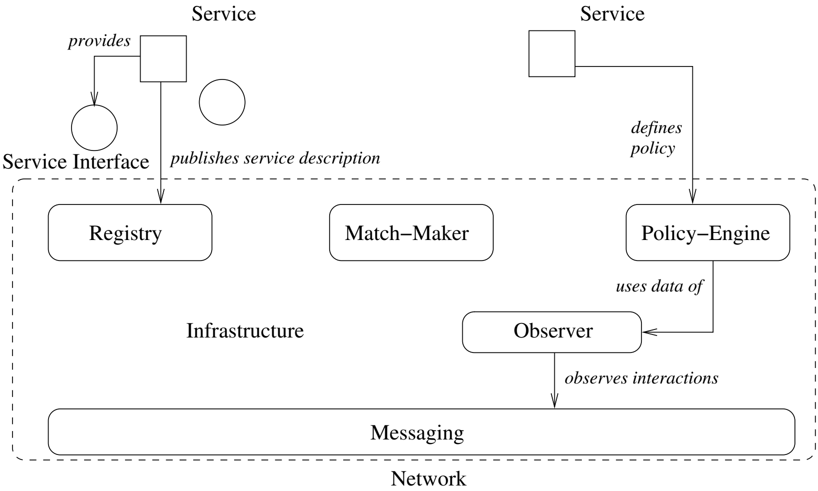
What components are required to base Service Selection on Dynamic QoS?



STRL

Service Architecture

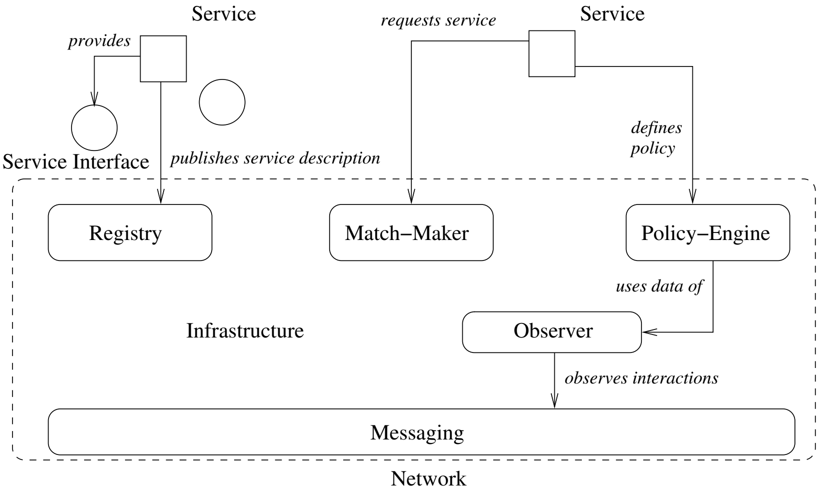
What components are required to base Service Selection on Dynamic QoS?



STRL

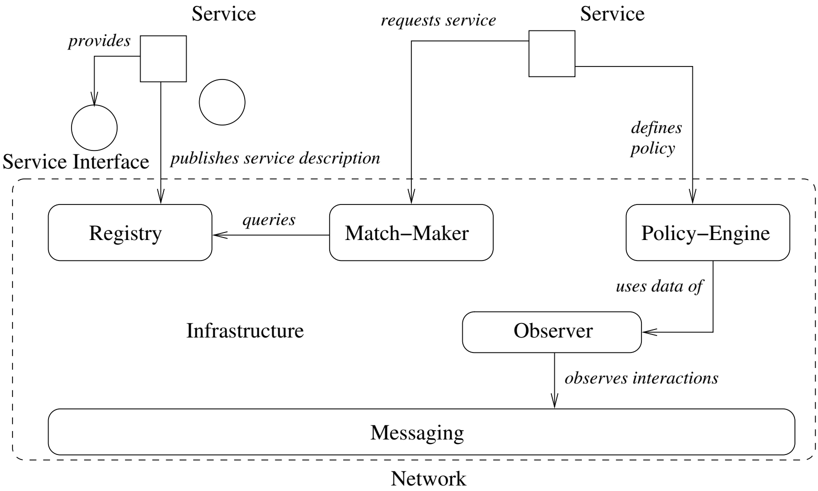
Service Architecture

What components are required to base Service Selection on Dynamic QoS?



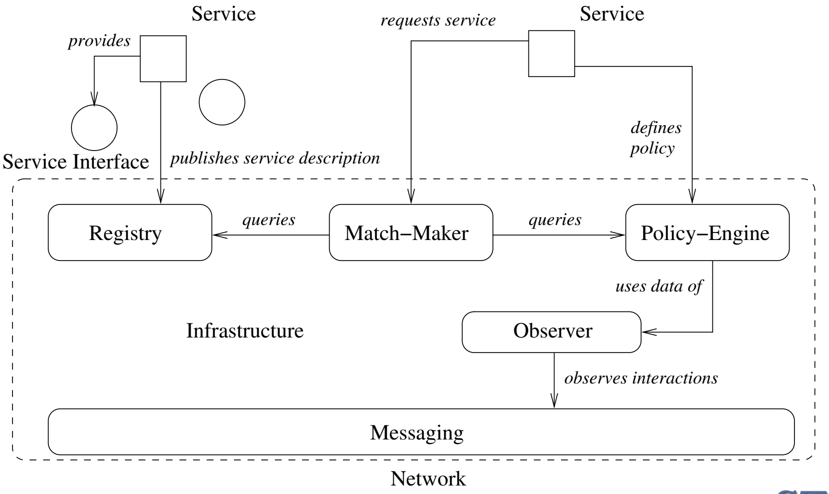
Service Architecture

What components are required to base Service Selection on Dynamic QoS?



Service Architecture

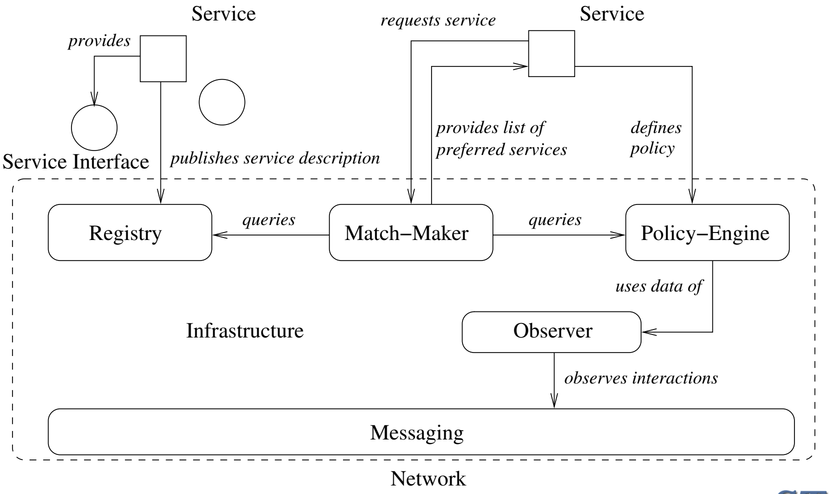
What components are required to base Service Selection on Dynamic QoS?



STRL

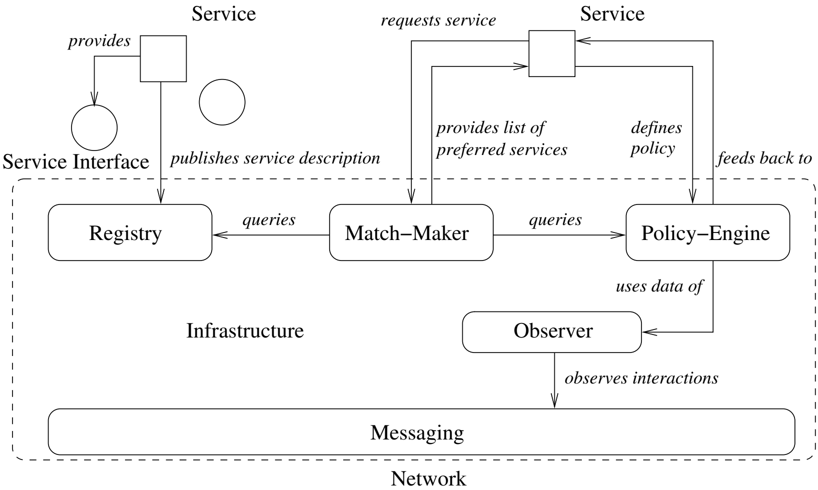
Service Architecture

What components are required to base Service Selection on Dynamic QoS?



Service Architecture

What components are required to base Service Selection on Dynamic QoS?



Policy & Preferences

How can we define subjective preferences as policy?

- Previous work on access control policies:
  - expressed in terms of rules.
  - rules map from observed behaviours to access decisions.
  - rules are composed into policies.
  - policies can change dynamically over time/ with events.
- This was extended with the notion of:
  - *mutable* attributes (Park 2004 [2])
  - pre, post and ongoing update *actions* (Sandhu 2004 [1])
- We express a preference rule as:

when *b* [**increase** | **decrease**] **preference**
in *s* [**little** | **medium** | **strong**]

Policy & Preferences

How can we define subjective preferences as policy?

- Previous work on access control policies:
  - expressed in terms of rules.
  - rules map from observed behaviours to access decisions.
  - rules are composed into policies.
  - policies can change dynamically over time/ with events.
- This was extended with the notion of:
  - *mutable* attributes (Park 2004 [2])
  - pre, post and ongoing update *actions* (Sandhu 2004 [1])
- We express a preference rule as:

when b **[increase | decrease] preference**
in s **[little | medium | strong]**

Policy & Preferences

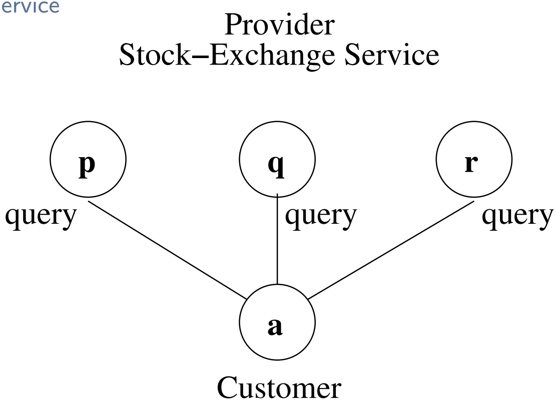
How can we define subjective preferences as policy?

- Previous work on access control policies:
  - expressed in terms of rules.
  - rules map from observed behaviours to access decisions.
  - rules are composed into policies.
  - policies can change dynamically over time/ with events.
- This was extended with the notion of:
  - *mutable* attributes (Park 2004 [2])
  - pre, post and ongoing update *actions* (Sandhu 2004 [1])
- We express a preference rule as:

when b [**increase** | **decrease**] **preference**
in s [**little** | **medium** | **strong**]

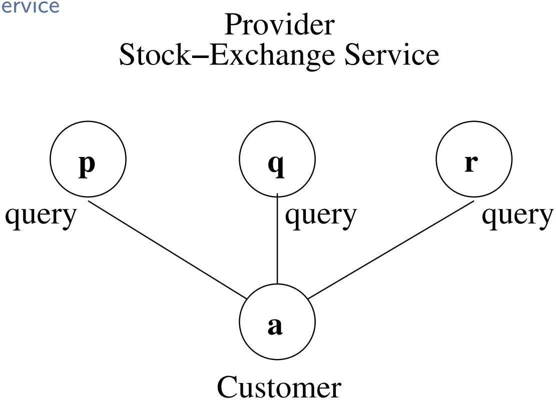
Example

A Stock-Quote Service



Example

A Stock-Quote Service



Customer **a** registers the following policy:

```
scope ([#a],[#p,#q,#r],[#query]) : {
  new ConsumerPolicy()
}
```

Example

Defining the policy

```
policy ConsumerPolicy {
  require Function int rt(Subject, Object, Action)
  define static int little = 1 /* ... */
  define S.O.A.pref = 0
  define Update incr(int x, int y) { x := x + y }

  perform incr(S.O.A.pref, medium)
  once after (S,O,A)
  when 10: (always rt(S,O,A) <= 2)

  perform decr(S.O.A.pref, little)
  once after (S,O,A)
  when 10: (sometime rt(S,O,A) >= 5)

  perform decr(S.O.A.pref, strong)
  after (S,O,A)
  when 0: (rt(S,O,A) >= 10)
}
```

Example

A Stock-Quote Service – Evaluation of the Policy

For simplicity we only look at provider **p**:

request	1	2	3	4	5	6	7	8	9	10	11	12	...
rt/sec	11	2	2	1	1	1	1	2	2	2	2	5	...
#a.#p.pref	0	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-1	-2

No policy rule fires

Example

A Stock-Quote Service – Evaluation of the Policy

For simplicity we only look at provider **p**:

request	1	2	3	4	5	6	7	8	9	10	11	12	...
rt/sec	11	2	2	1	1	1	1	2	2	2	2	5	...
#a.#p.pref	0	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-1	-2

```
  perform decr(S.O.A.pref, strong)
  after (S,O,A)
  when 0: (rt(S,O,A) >= 10)
```

Example

A Stock-Quote Service – Evaluation of the Policy

For simplicity we only look at provider **p**:

request	1	2	3	4	5	6	7	8	9	10	11	12	…
rt/sec	11	2	2	1	1	1	1	2	2	2	2	5	…
#a.#p.pref	0	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-1	-2

```
  perform decr(S.O.A.pref, strong)
  after (S,O,A)
  when 0: (rt(S,O,A) >= 10)
```

Example

A Stock-Quote Service – Evaluation of the Policy

For simplicity we only look at provider **p**:

request	1	2	3	4	5	6	7	8	9	10	11	12	...
rt/sec	11	2	2	1	1	1	1	2	2	2	2	5	...
#a.#p.pref	0	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-1	-2

No policy rule fires

Example

A Stock-Quote Service – Evaluation of the Policy

For simplicity we only look at provider **p**:

request	1	2	3	4	5	6	7	8	9	10	11	12	...
rt/sec	11	2	2	1	1	1	1	2	2	2	2	5	...
#a.#p.pref	0	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-1	-2

No policy rule fires

Example

A Stock-Quote Service – Evaluation of the Policy

For simplicity we only look at provider **p**:

request	1	2	3	4	5	6	7	8	9	10	11	12	...
rt/sec	11	2	2	1	1	1	1	2	2	2	2	5	...
#a.#p.pref	0	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-1	-2

No policy rule fires

Example

A Stock-Quote Service – Evaluation of the Policy

For simplicity we only look at provider **p**:

request	1	2	3	4	5	6	7	8	9	10	11	12	...
rt/sec	11	2	2	1	1	1	1	2	2	2	2	5	...
#a.#p.pref	0	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-1	-2

No policy rule fires

Example

A Stock-Quote Service – Evaluation of the Policy

For simplicity we only look at provider **p**:

request	1	2	3	4	5	6	7	8	9	10	11	12	...
rt/sec	11	2	2	1	1	1	1	2	2	2	2	5	...
#a.#p.pref	0	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-1	-2

No policy rule fires

Example

A Stock-Quote Service – Evaluation of the Policy

For simplicity we only look at provider **p**:

request	1	2	3	4	5	6	7	8	9	10	11	12	...
rt/sec	11	2	2	1	1	1	1	2	2	2	2	5	...
#a.#p.pref	0	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-1	-2

No policy rule fires

Example

A Stock-Quote Service – Evaluation of the Policy

For simplicity we only look at provider **p**:

request	1	2	3	4	5	6	7	8	9	10	11	12	...
rt/sec	11	2	2	1	1	1	1	2	2	2	2	5	...
#a.#p.pref	0	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-1	-2

No policy rule fires

Example

A Stock-Quote Service – Evaluation of the Policy

For simplicity we only look at provider **p**:

request	1	2	3	4	5	6	7	8	9	10	11	12	...
rt/sec	11	2	2	1	1	1	1	2	2	2	2	5	...
#a.#p.pref	0	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-1	-2

No policy rule fires

Example

A Stock-Quote Service – Evaluation of the Policy

For simplicity we only look at provider **p**:

request	1	2	3	4	5	6	7	8	9	10	11	12	…
rt/sec	11	2	2	1	1	1	1	2	2	2	2	5	…
#a.#p.pref	0	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-1	-2

No policy rule fires

Example

A Stock-Quote Service – Evaluation of the Policy

For simplicity we only look at provider **p**:

request	1	2	3	4	5	6	7	8	9	10	11	12	…
rt/sec	11	2	2	1	1	1	1	2	2	2	2	5	…
#a.#p.pref	0	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-1	-2

No policy rule fires

Example

A Stock-Quote Service – Evaluation of the Policy

For simplicity we only look at provider **p**:

request	1	2	3	4	5	6	7	8	9	10	11	12	...
rt/sec	11	2	2	1	1	1	1	2	2	2	2	5	...
#a.#p.pref	0	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-1	-2

No policy rule fires

Example

A Stock-Quote Service – Evaluation of the Policy

For simplicity we only look at provider **p**:

request	1	2	3	4	5	6	7	8	9	10	11	12	...
rt/sec	11	2	2	1	1	1	1	2	2	2	2	5	...
#a.#p.pref	0	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-1	-2

No policy rule fires

Example

A Stock-Quote Service – Evaluation of the Policy

For simplicity we only look at provider **p**:

request	1	2	3	4	5	6	7	8	9	10	11	12	...
rt/sec	11	2	2	1	1	1	1	2	2	2	2	5	...
#a.#p.pref	0	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-1	-2

No policy rule fires

Example

A Stock-Quote Service – Evaluation of the Policy

For simplicity we only look at provider **p**:

request	1	2	3	4	5	6	7	8	9	10	11	12	…
rt/sec	11	2	2	1	1	1	1	2	2	2	2	5	…
#a.#p.pref	0	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-1	-2

No policy rule fires

Example

A Stock-Quote Service – Evaluation of the Policy

For simplicity we only look at provider **p**:

request	1	2	3	4	5	6	7	8	9	10	11	12	...
rt/sec	11	2	2	1	1	1	1	2	2	2	2	5	...
#a.#p.pref	0	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-1	-2

No policy rule fires

Example

A Stock-Quote Service – Evaluation of the Policy

For simplicity we only look at provider **p**:

request	1	2	3	4	5	6	7	8	9	10	11	12	...
rt/sec	11	2	2	1	1	1	1	2	2	2	2	5	...
#a.#p.pref	0	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-1	-2

No policy rule fires

Example

A Stock-Quote Service – Evaluation of the Policy

For simplicity we only look at provider **p**:

request	1	2	3	4	5	6	7	8	9	10	11	12	...
rt/sec	11	2	2	1	1	1	1	2	2	2	2	5	...
#a.#p.pref	0	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-1	-2

No policy rule fires

Example

A Stock-Quote Service – Evaluation of the Policy

For simplicity we only look at provider **p**:

request	1	2	3	4	5	6	7	8	9	10	11	12	...
rt/sec	11	2	2	1	1	1	1	2	2	2	2	5	...
#a.#p.pref	0	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-1	-2

No policy rule fires

Example

A Stock-Quote Service – Evaluation of the Policy

For simplicity we only look at provider **p**:

request	1	2	3	4	5	6	7	8	9	10	11	12	…
rt/sec	11	2	2	1	1	1	1	2	2	2	2	5	…
#a.#p.pref	0	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-1	-2

```
  perform incr(S.O.A.pref, medium)
  once after (S,O,A)
  when 10: (always rt(S,O,A) <= 2)
```

Example

A Stock-Quote Service – Evaluation of the Policy

For simplicity we only look at provider **p**:

request	1	2	3	4	5	6	7	8	9	10	11	12	…
rt/sec	11	2	2	1	1	1	1	2	2	2	2	5	…
#a.#p.pref	0	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-1	-2

```
  perform incr(S.O.A.pref, medium)
  once after (S,O,A)
  when 10: (always rt(S,O,A) <= 2)
```

Example

A Stock-Quote Service – Evaluation of the Policy

For simplicity we only look at provider **p**:

request	1	2	3	4	5	6	7	8	9	10	11	12	...
rt/sec	11	2	2	1	1	1	1	2	2	2	2	5	...
#a.#p.pref	0	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-1	-2

```
  perform decr(S.O.A.pref, little)
  once after (S,O,A)
  when 10: (sometime rt(S,O,A) >= 5)
```

Example

A Stock-Quote Service – Evaluation of the Policy

For simplicity we only look at provider **p**:

request	1	2	3	4	5	6	7	8	9	10	11	12	...
rt/sec	11	2	2	1	1	1	1	2	2	2	2	5	...
#a.#p.pref	0	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-1	-2

```
perform decr(S.O.A.pref, little)
once after (S,O,A)
when 10: (sometime rt(S,O,A) >= 5)
```

Example

A Stock-Quote Service – Evaluation of the Policy

For simplicity we only look at provider **p**:

request	1	2	3	4	5	6	7	8	9	10	11	12	...
rt/sec	11	2	2	1	1	1	1	2	2	2	2	5	...
#a.#p.pref	0	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-1	-2

- Preferences are updated continuously based on experience.
- Rules define:
  - The time/context of observations are defined.
  - The condition for updates as a behaviour of these observations.
  - The effect that an update has on policy attributes.
- The policy attribute (e.g. #a.O.#query.pref) can be seen as an ordering of service providers O $\in$ [#p, #q, #r].

Conclusion

The nice ...

- Architecture
  - Service Selection is made based on *dynamic* QoS attributes.
  - Support is provided at the infrastructure level, viz.
    - Observations can be made "objectively".
    - More observations can be taken into account (if authorisation constraints permit).
    - Policies are continuously evaluated.
    - Service Selection is transparent to the consumer.
    - Feedback to consumer on (significant) changes.
- Policies
  - Formal semantics: Validation & Verification of properties.
  - Define QoS declaratively.
  - QoS is defined *subjectively* by each consumer taking past experiences into account.
  - Integrate nicely with access control policies defining who can observe dynamic attributes of a consumers interactions.

Conclusion

.. and ugly

- Overhead on the infrastructure.
- Communication of non-local observations diminishes bandwidth.
- Missing methodology to define preference rules and update actions.
- Validation that the policy captures the intent.
- More potential for policy conflicts.
- Update actions complicate semantics.

Questions?

... are appreciated!

Contact:

Helge Janicke
(heljanic@dmu.ac.uk)

Monika Solanki
(monika@doc.ic.ac.uk)

J. Park and R. S. Sandhu.
The ucon$_{\text{abc}}$ usage control model.
ACM Trans. Inf. Syst. Secur., 7(1):128–174, 2004.

J. Park, X. Zhang, and R. S. Sandhu.
Attribute mutability in usage control.
In C. Farkas and P. Samarati, editors, *Proceedings of IFIP TC11/WG 11.3 Eighteenth Annual Conference on Data and Applications Security*, pages 15–29, Sitges, Catalonia, Spain, July 2004. Kluwer.