Policy-driven Reflective Enforcement of Security Policies

Ian Welch and Fan Lu

Overview

Want to separate out enforcement code from other code

In context of Java, allow application-level enforcement to be handled as transparently as system-level enforcement

But solution requires two policy files, one of which is imperative

Want to write one policy using a DSL and have everything else taken care of automatically

Example – chat client

IRC chat client (3rd party component).

Standalone or used as part of another application.

Aim is to enforce a local policy.

Restrict access by user to given chat rooms or channels.

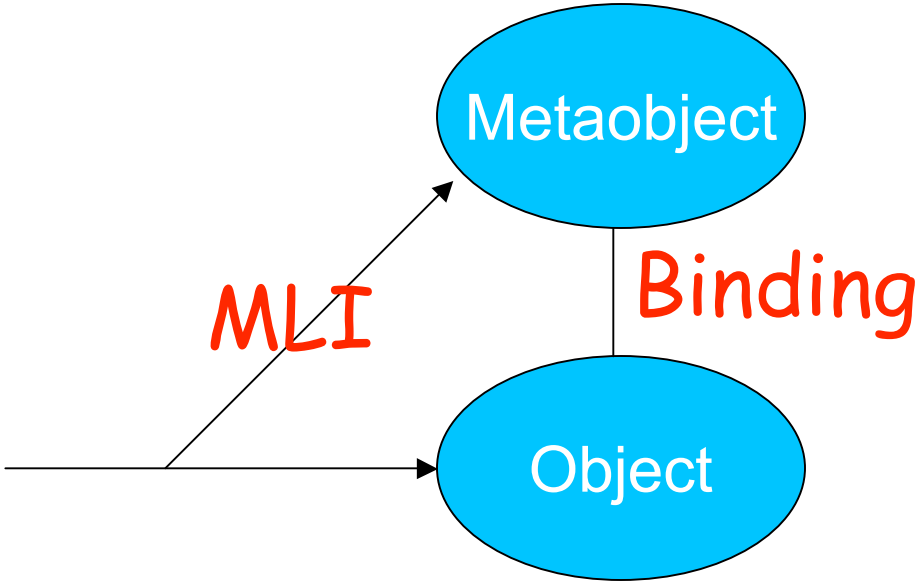
Security implemented using a Kava metaobject protocol.

Kava

Metaobject redefines object behaviour.

Binding achieved at loadtime.

Similar to Aspects except crosscutting code is distributed across multiple metaobjects.



Reflective enforcement

```
public class Lirc {...};
```

Chat client

```
(new Lirc).createChannel(channelName);
```

```
public class EnforcementMetaObject extends MetaObject {
  ...
  public void beforeExecuteMethod(...) {
    ...
    sm.checkPermission(new ChannelPermission(
        joinChannel, "join");
  }}
```

```
<intercept><execute><cname>Lirc</cname>
<method>createChannel</method></execute></intercept>
```

```
grant signedby WhiteHat {
  permission ChannelPermission allowedChannel, join;
}
```

Reflective enforcement

Chat client

```
public class Lirc {...};
```

Program using client

```
(new Lirc).createChannel(channelName);
```

```
public class EnforcementMetaObject extends MetaObject {
  ...
  public void beforeExecuteMethod(...) {
    ...
    sm.checkPermission(new ChannelPermission(
        joinChannel, "join");
  }}
```

```
<intercept><execute><cname>Lirc</cname>
<method>createChannel</method></execute></intercept>
```

```
grant signedby WhiteHat {
  permission ChannelPermission allowedChannel, join;
}
```

Reflective enforcement

Chat client

```
public class Lirc {...};
```

Program using client

```
(new Lirc).createChannel(channelName);
```

Metaobject

```
public class EnforcementMetaObject extends MetaObject {
  ...
  public void beforeExecuteMethod(...) {
    ...
    sm.checkPermission(new ChannelPermission(
        joinChannel, "join");
  }}
```

```
<intercept><execute><cname>Lirc</cname>
<method>createChannel</method></execute></intercept>
```

```
grant signedby WhiteHat {
  permission ChannelPermission allowedChannel, join;
}
```

Reflective enforcement

```
public class Lirc {...};
```

Chat client

```
(new Lirc).createChannel(channelName);
```

Program using client

```
public class EnforcementMetaObject extends MetaObject {
  ...
  public void beforeExecuteMethod(...) {
    ...
    sm.checkPermission(new ChannelPermission(
        joinChannel, "join");
  }}
```

Metaobject

```
<intercept><execute><cname>Lirc</cname>
<method>createChannel</method></execute><binding cpt>
```

Binding specification

```
grant signedby WhiteHat {
  permission ChannelPermission allowedChannel, join;
}
```

Reflective enforcement

```
public class Lirc {...};
```

Chat client

```
(new Lirc).createChannel(channelName);
```

Program using client

```
public class EnforcementMetaObject extends MetaObject {
  ...
  public void beforeExecuteMethod(...) {
    ...
    sm.checkPermission(new ChannelPermission(
        joinChannel, "join");
  }}
```

Metaobject

```
<intercept><execute><cname>Lirc</cname>
<method>createChannel</method></execute><metaobject>
```

Binding specification

```
grant signedby WhiteHat {
  permission ChannelPermission allowedChannel, join;
}
```

Java security policy

Interdependencies

```
public class Lirc {...};
```

Chat client

```
(new Lirc).createChannel(channelName);
```

Program using client

```
public class EnforcementMetaObject extends MetaObject {
  ...
  public void beforeExecuteMethod(...) {
    ...
    sm.checkPermission(new ChannelPermission(
        joinChannel, "join");
  }}
```

```
<intercept><execute><cname>Lirc</cname>
<method>createChannel</method></execute></intercept>
```

```
grant signedby WhiteHat {
  permission ChannelPermission allowedChannel, join;
};
```

Choice of permissions related to abstract resource

Interdependencies

```
public class Lirc {...};                                    Chat client
```

```
(new Lirc).createChannel(channelName);                      Program using client
```

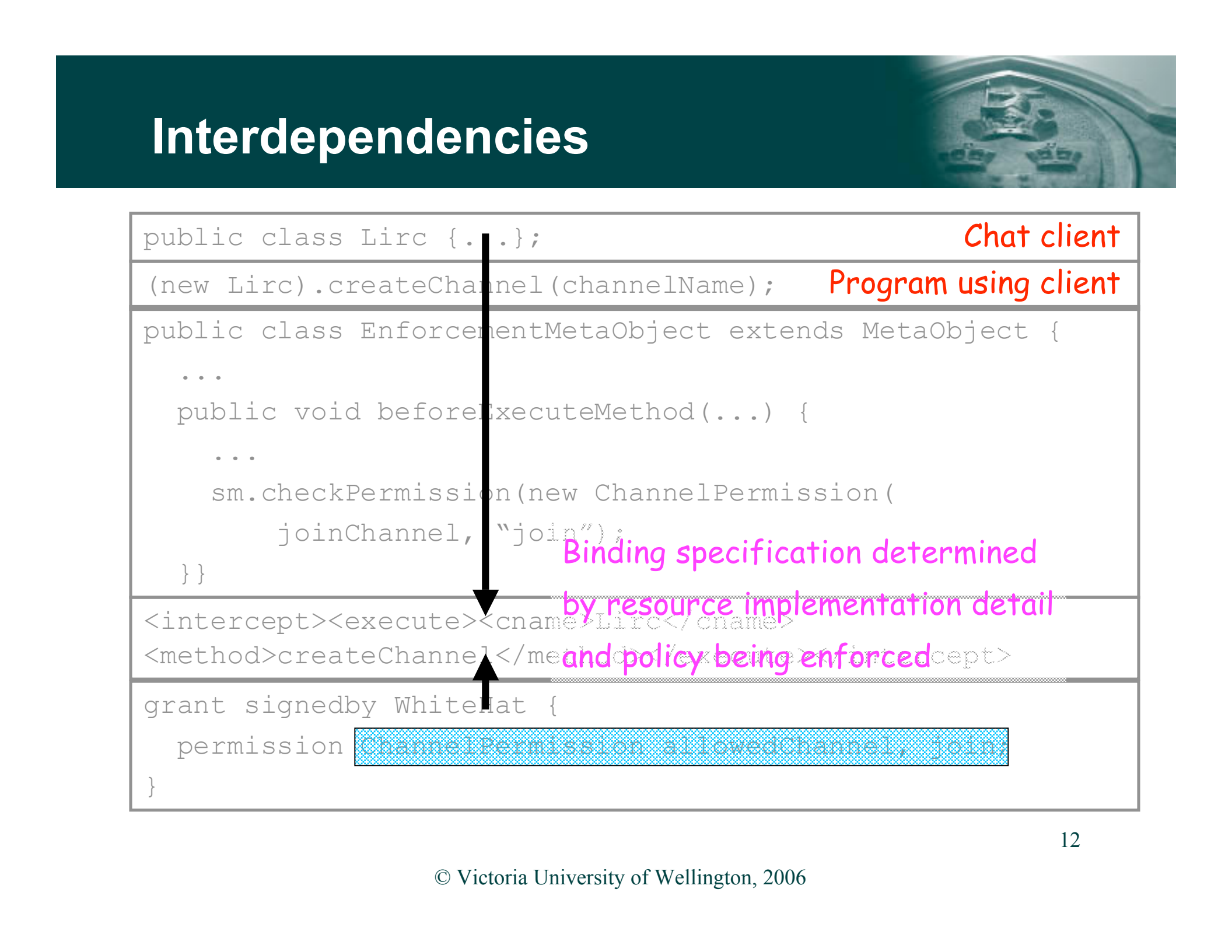
```
public class EnforcementMetaObject extends MetaObject {
   ...
   public void beforeExecuteMethod(...) {
     ...
     sm.checkPermission(new ChannelPermission(
                                  "join");
   }}
```

Choice of metaobject related to abstract resource that is subject of policy

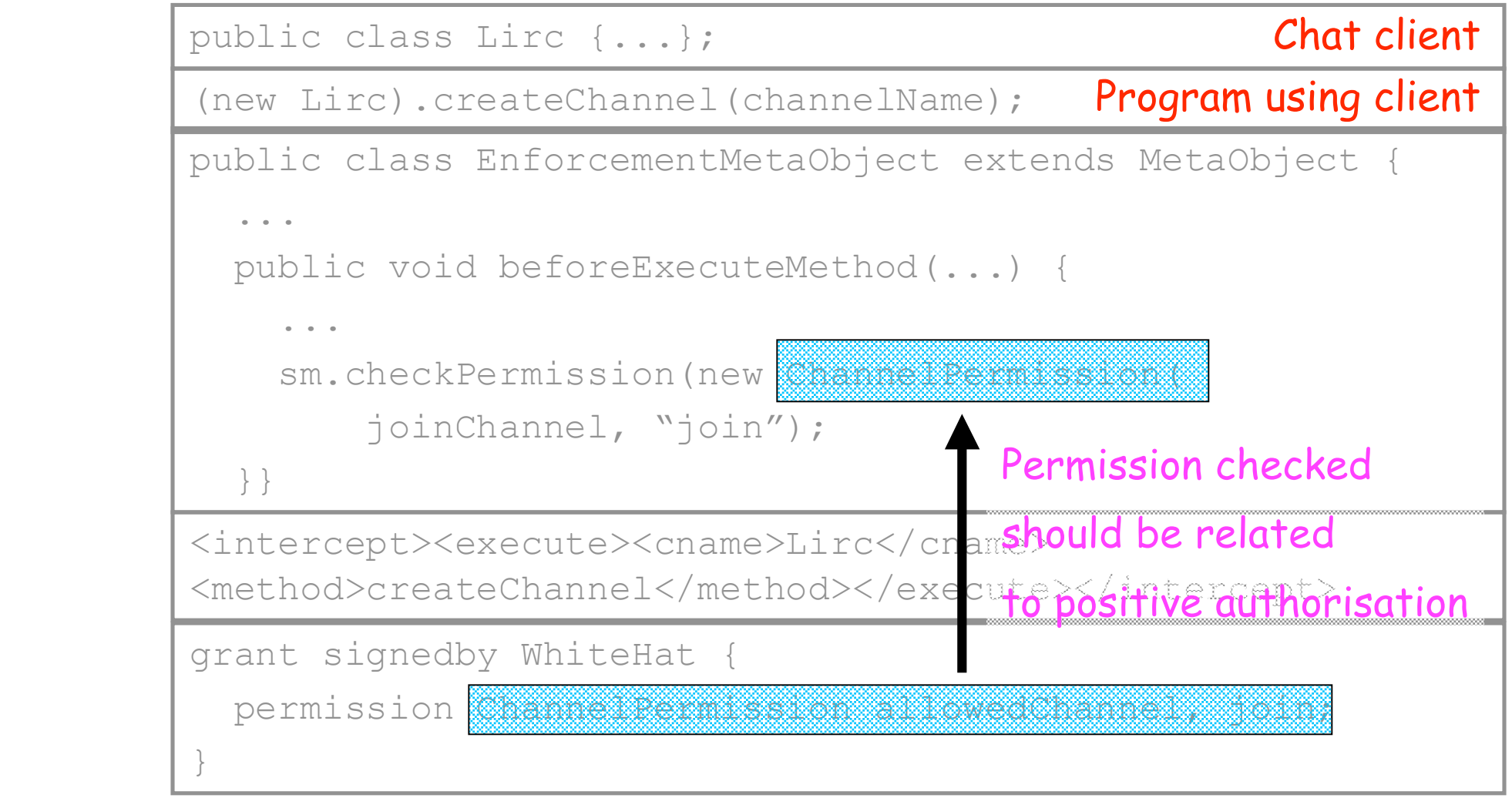
```
<intercept><execute><cname>Lirc</cname>
<method>createChannel</method></execute></intercept>
```

```
grant signedby WhiteHat {
  permission ChannelPermission allowedChannel, join;
}
```

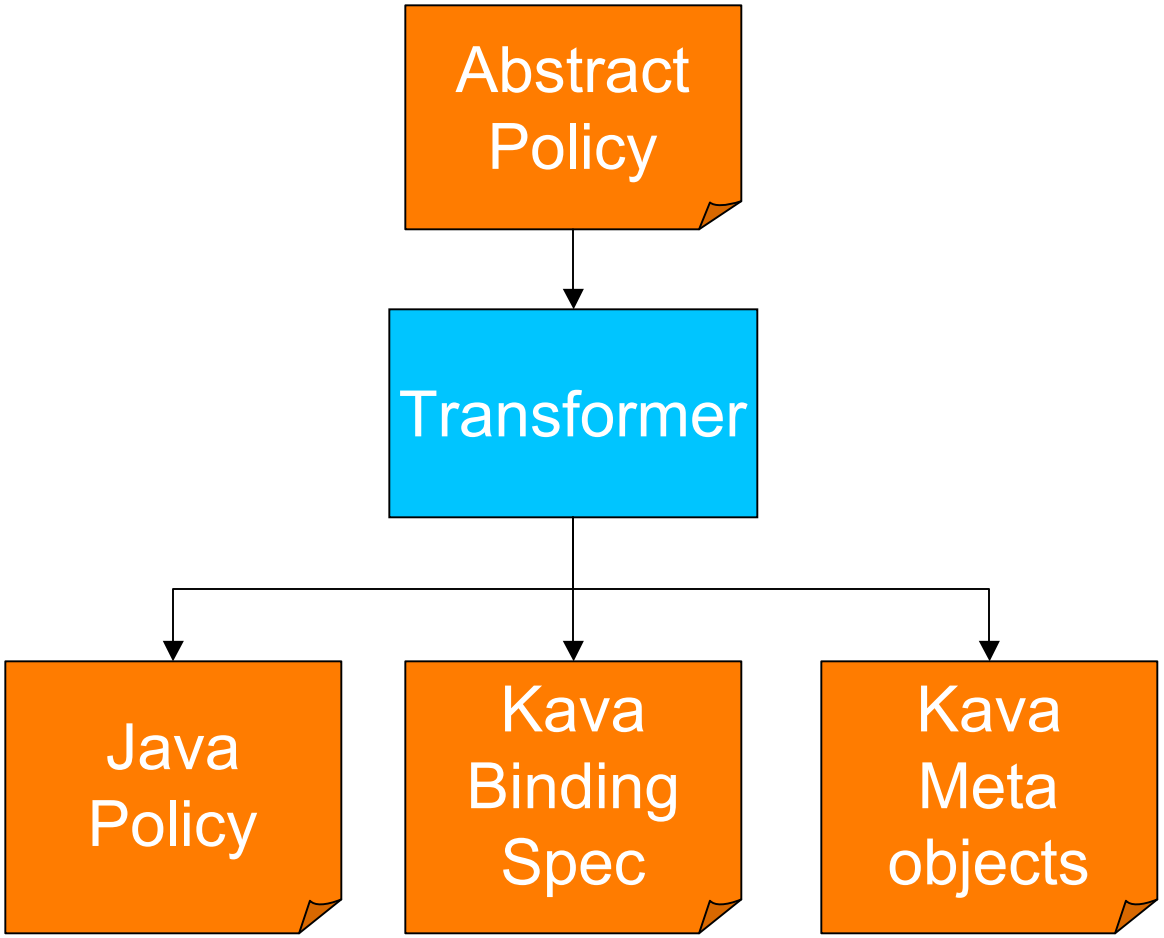
Interdependencies



Interdependencies



Model



Ponder policy language

Designed to specify wide range of policies.

Declarative rather than imperative.

Positive, negative and constraint based policies.

Envisaged for enterprise-level enforcement so used LDAP to store policies and perform mappings.

```
inst auth+ rpc_chat {
  subject /staff/securityAdmin ;
  target <Channel> /channels/support ;
  action join, speak ;
}
```

One policy file!

Issues

Subject

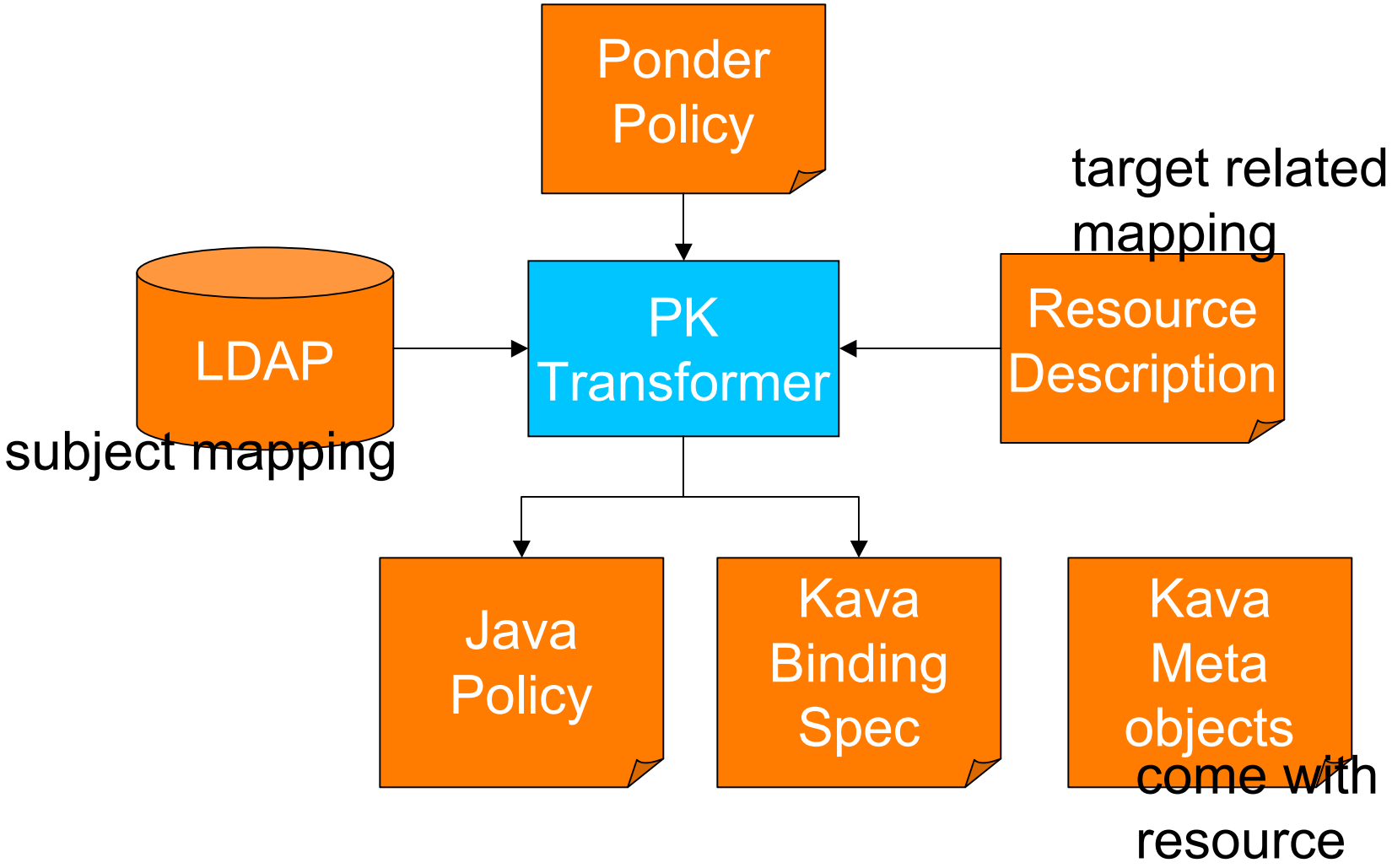
- Ponder objects -> Java protection domains

Targets and actions

- Ponder target is an object, actions relate to interface
- Java permissions relate to set of classes
- Java targets and actions optional

Don’t want to hardcode mappings to avoid rewriting transformer

Solution



Resource description

Defines vocabulary for Ponder policy

Mappings for application and system resources.

Mapping rules:

- subject-domain ⇒ protection domain
- target-type ⇒ Java permission
- target-domain & actions ⇒ permission parameters
- target-type ⇒ binding specification/choice of metaobjects

Results

Benefits

- Relationships exposed and localised in one place (almost).
- Policies expressed using high-level language-neutral application abstractions.

Issues

- Metaobject generation
- What happens with overlapping Ponder policies?
- How to implement constraints? negative authorisations?
- Is it really easier to use? DSL question.