D\textsuperscript{2}AL

A Design-based Distribution Aspect Language

Ulrich Becker
University of Erlangen-Nuremberg
Department of Computer Science IV
Introduction

**Motivation**
- optimized distribution required for efficient distributed applications
- software developer should have explicit control

**$D^2AL$ is an aspect language for object distribution**
- improve application performance through...
  - collocation of collaborating objects
  - migration of objects
  - replication of immutable objects

**$D^2AL$ is based on the UML design model of the application**
- design contains distribution-relevant information that is lost in implementation
  - static structure: associations
  - behaviour: state machines
Collaborations (1)

Collaborations are the basis for object collocation

- collaborations are pairs of objects
- related by an association

EMail class diagram

```
MailAccount sender EMail content EMailContent receiver
```

\(D^2AL\) statement for collocation

```
collaboration EMailCluster {
  link content;
  distribution collocation;
}
```

- collaborations can be limited to abstract states
Collaborations (2)

**EMail: State diagram**

1. **a:**MailAccount \( \rightarrow \) :EMail \( \rightarrow \) b:**MailAccount
   - sender
   - Node A
   - ** receiver
   - Node B

2. **a:**MailAccount \( \rightarrow \) :EMail \( \rightarrow \) b:**MailAccount
   - sender
   - Node A
   - ** receiver
   - Node B

**Send**

- send failed
- send successful
- Undeliverable
- Delivered
Replication

- Abstract states can be declared *replicable*
- Decision whether to replicate is left to runtime system

$D^2AL$ statement for replication

```plaintext
states EMail {
  replicable {Undeliverable, Delivered};
}
```

1. Node A
   - a:MailAccount (sender)
   - :EMail
   - Node B
     - b:MailAccount (receiver)

2. Node A
   - a:MailAccount (sender)
   - :EMail
   - Node B
     - b:MailAccount (receiver)
     - :EMail
Weaver

Weaved code contains classes and statements to...
- maintain design information at runtime (associations, abstract states)
- make distribution decisions based on this information

Weaver

D²AL specification

```java
import java.util.Date;

public class DMail {
    private EMailContent content;

    public void send() {
        // do something
    }
}
```

Java source

```java
public class EMail {
    private EMailState myState;

    public void send() {
        myState.send();
    }
}
```

D²AL weaver
Conclusion

Advantages of $D^2AL$
- relationships and abstract states from the model can be used
  - increased expressiveness
  - no need to manually maintain abstract state in implementation
- $D^2AL$ specification is less prone to implementation changes

Current work
- prototype implementation

Future work
- validate suitability of design-based distribution specification
- consider other model-elements, e.g. interaction diagrams