Towards an Automated Pattern Selection Procedure in Software Models

Alexander van den Berghe, Jan Van Haaren,
Stefan Van Baelen, Yolande Berbers and Wouter Joosen

{firstname.lastname}@cs.kuleuven.be
IBBT-DistriNet, Department of Computer Science, KU Leuven
Celestijnenlaan 200A, 3001 Leuven, Belgium

22nd International Conference on Inductive Logic Programming
Monday September 17th 2012

KATHOLIEKE UNIVERSITEIT LEUVEN DistriNet RESEARCH GROUP

ibbt Security Department
Security, Privacy & Trust of E-*

Software (development) is becoming increasingly complex

Three main reasons:

1. Increased complexity of problems to be solved by software
2. Shift towards distributed software
3. Relatively long lifetime of software

Software patterns are used to manage this complexity:

- Selecting patterns is *hard*, *knowledge intensive* and *time-consuming*
- Instantiating patterns is *repetitive* and *error-prone*

We propose an automated approach to selecting software patterns with two contributions:

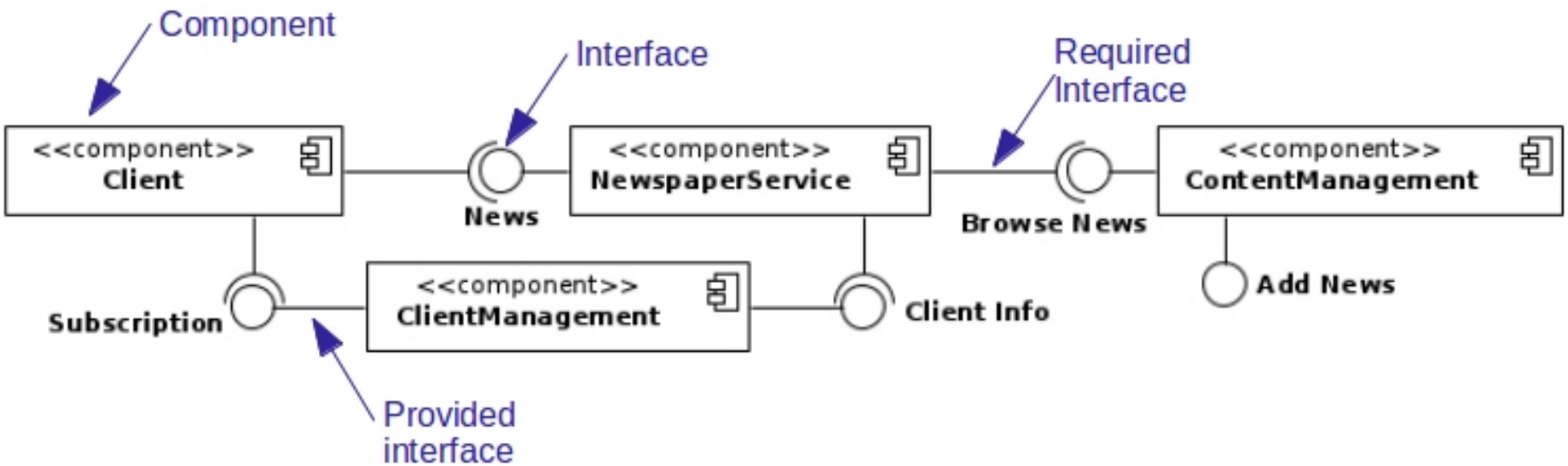
- Formal representation of software models and patterns
- Automatically learn roles for software models

Software development is gathering requirements and ensuring the system meets them

Achieved by defining a number of components which each:

- Satisfy a subset of the gathered requirements
- Offer functionality through one or more interfaces

Graphical models capture design decisions more formally



Excerpt of a digital news system's model

Software patterns provide established solutions to recurring design issues

1. **Name**

   Client-Dispatcher-Server

2. **Resolved design issue**

   Clients have to use services regardless the location of the servers

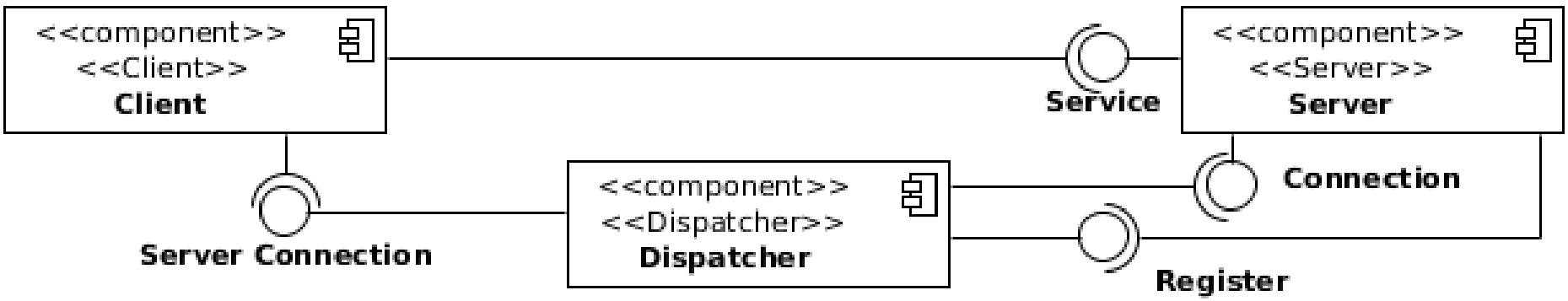
3. **Provided solution**

   Add a dispatcher as an intermediate layer between clients and servers

4. **Consequences**

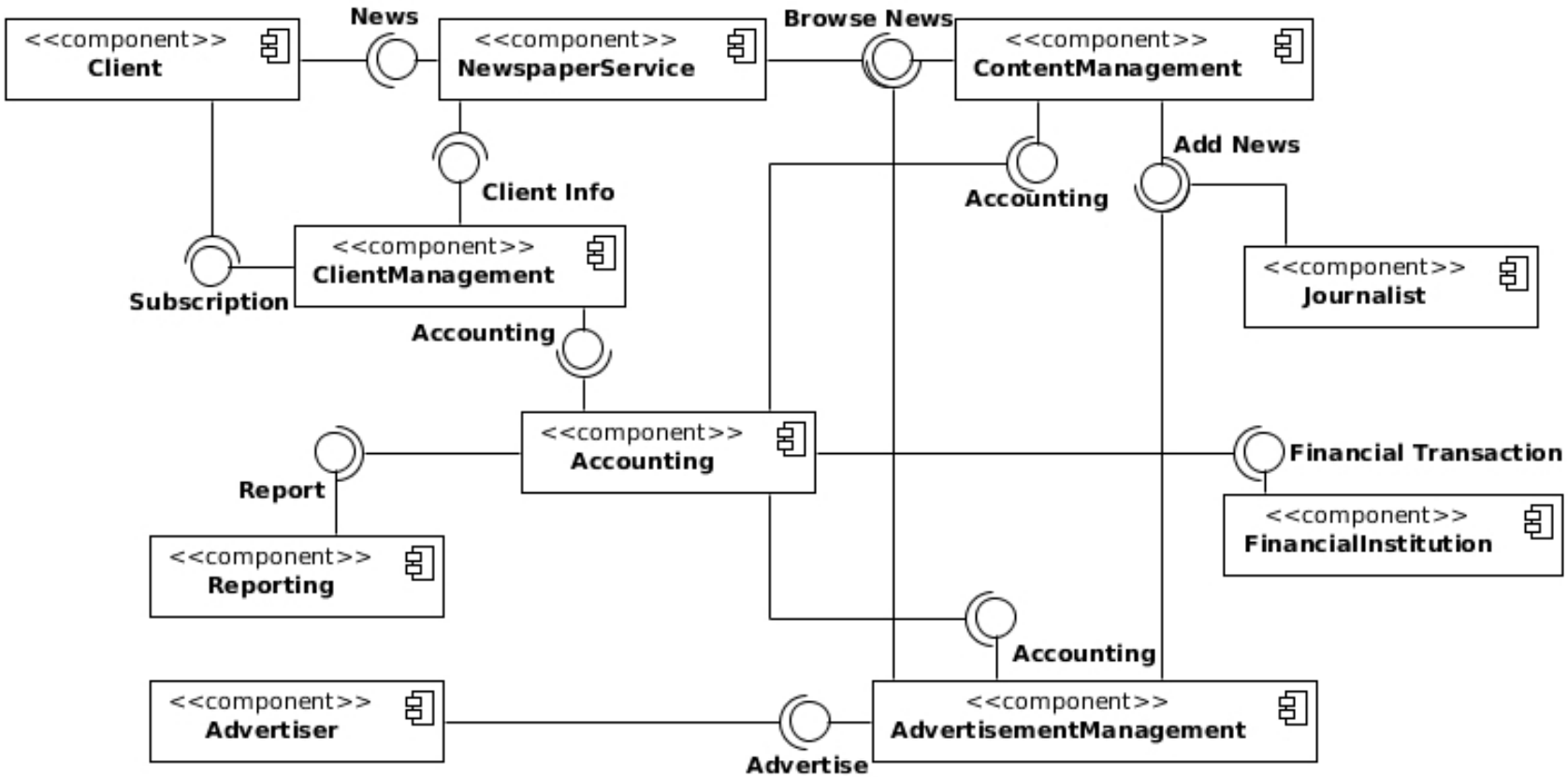
   Decouple clients and servers ✔
   Dispatcher is a possible bottleneck ✗

Graphical models capture the solution more formally



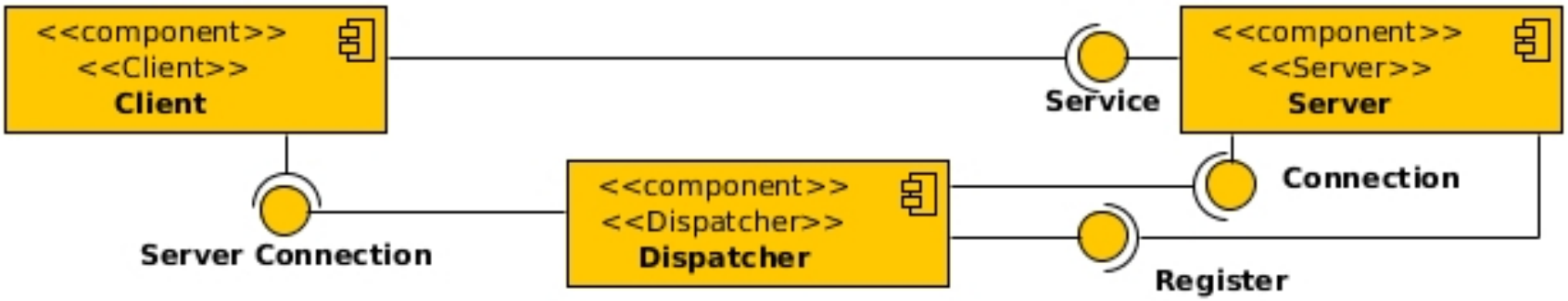
Client-Dispatcher-Server pattern

Given: a software model



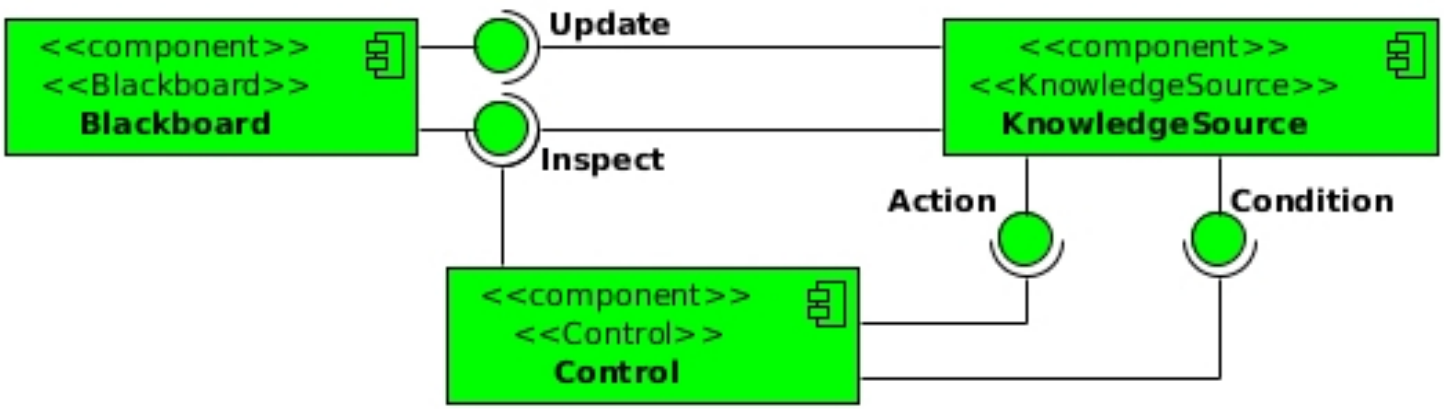
A digital news system

Given: a collection of software patterns



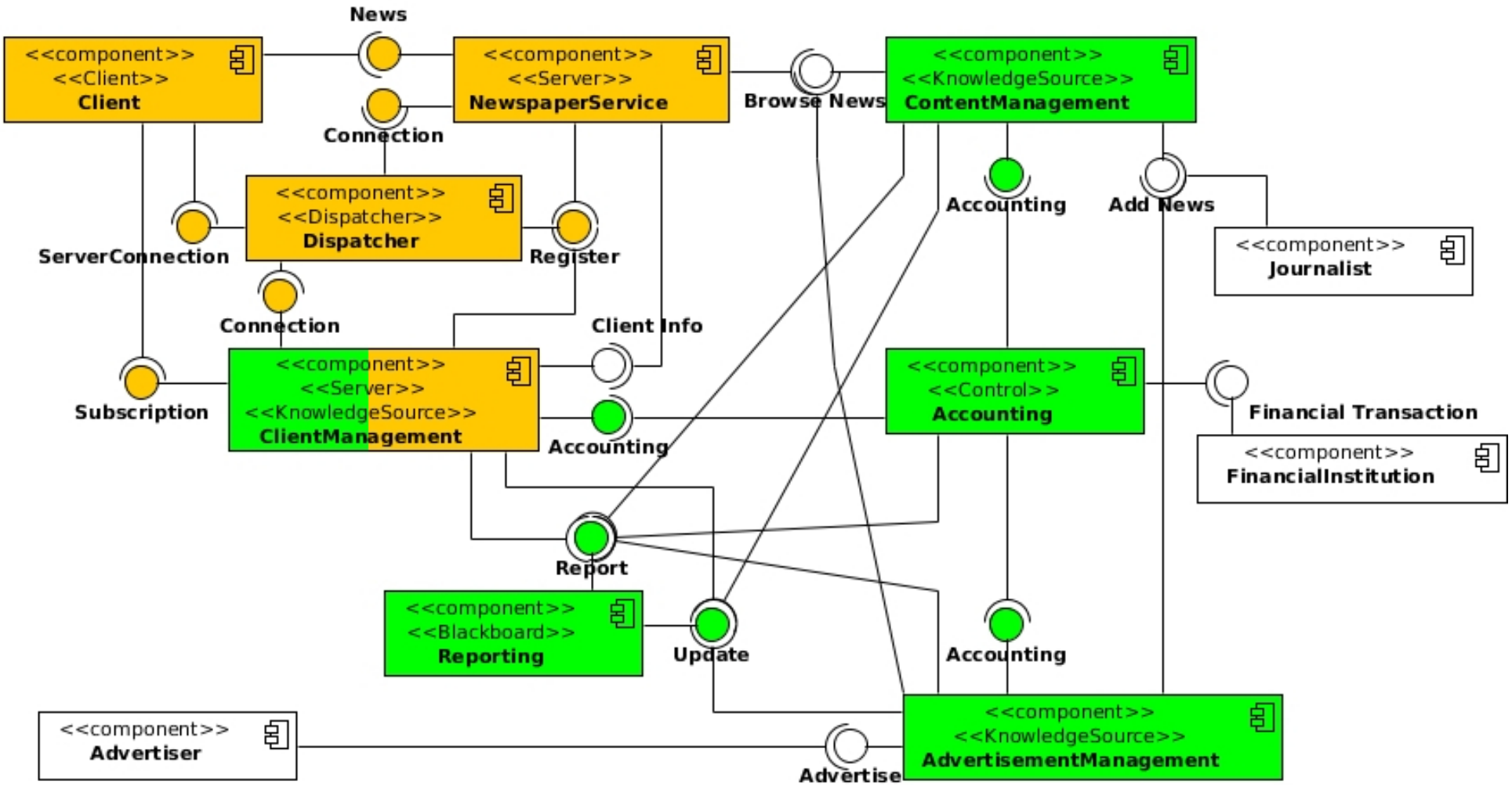
Client-Dispatcher-Server pattern

⋮ (other pattern models) ⋮



Blackboard pattern

Do: instantiate patterns in a software model



Patterns instantiated in the digital news system

Overview of our approach

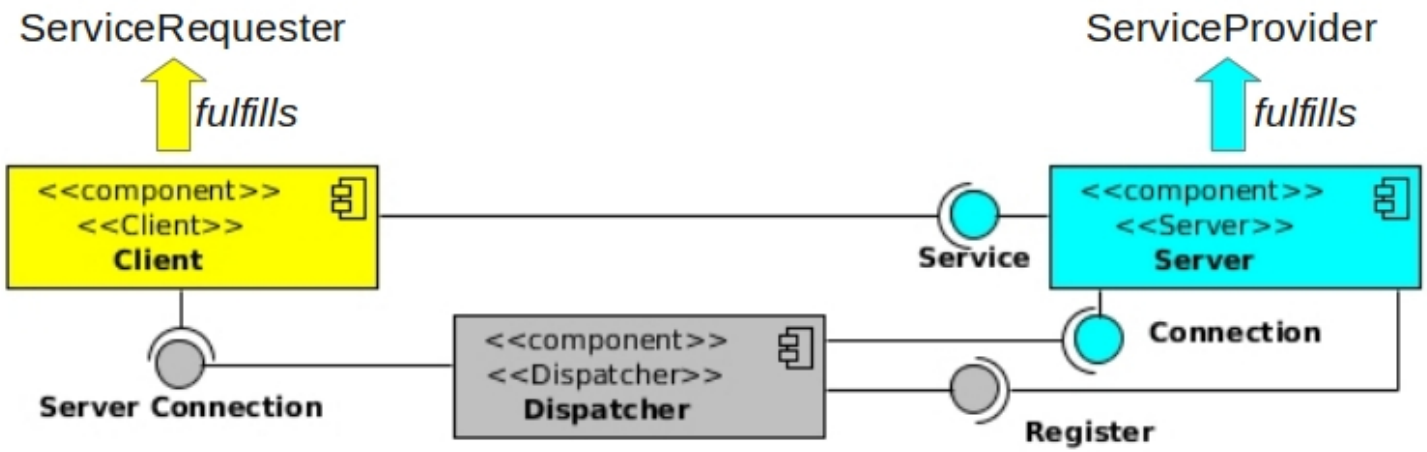
Observation: software models and patterns are highly relational

Our approach consists of three steps:

1. Represent the available patterns formally
2. Assign roles to the components in the software model
3. Select applicable patterns by leveraging the assigned roles

Step 1: Represent patterns formally

A software pattern is represented as a set of **primitives**
A primitive is a precisely defined building block that can **fulfill roles**

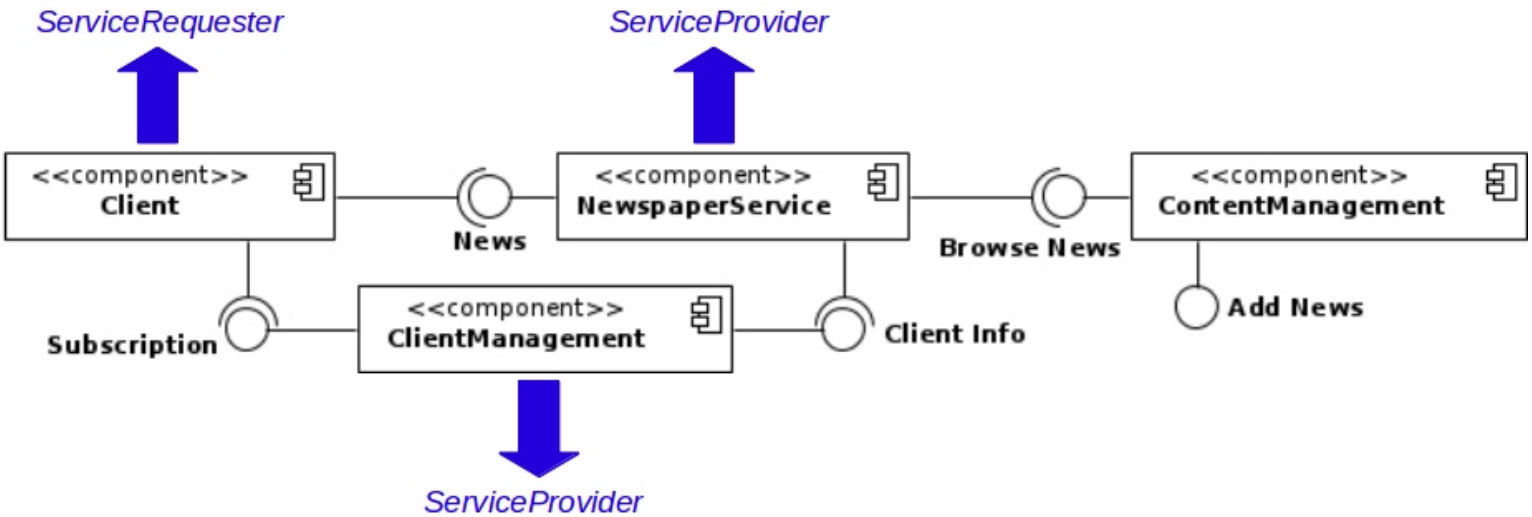


Client-Dispatcher-Pattern with three primitives

Implementation: Prolog knowledge base

```
pattern(clientDispatcherServer).
primitive(client).
primitive(server).
primitive(dispatcher).
role(serviceRequester).
role(serviceProvider).
fulfills(client, serviceRequester).
fulfills(server, serviceProvider).
used_in(client, clientDispatcherServer).
used_in(dispatcher, clientDispatcherServer).
used_in(server, clientDispatcherServer).
```

Step 2: Assign roles to components in the software model



Excerpt of digital news system's model with the assigned roles

Facts derived from assigned roles:

required_role(serviceRequester).

required_role(serviceProvider).

Step 2: Automatically assigning roles

Learning task:

A collective classification task on relational data

Given:

- Software model at hand (optionally with manually assigned roles)
- Software models with previously learned/assigned roles

Learn:

Roles of the components in the software model at hand

Step 2: Any relational learner can solve this learning task

We are working on a **proof-of-concept** using **kLog**

(`http://people.cs.kuleuven.be/~alexander.vandenberghe/arbps.html`)

What is kLog?

- Logical and relational language for kernel-based learning
- Builds upon several simple but powerful concepts
  (i.e., entity-relationship data modeling and graph kernels)

Why kLog?

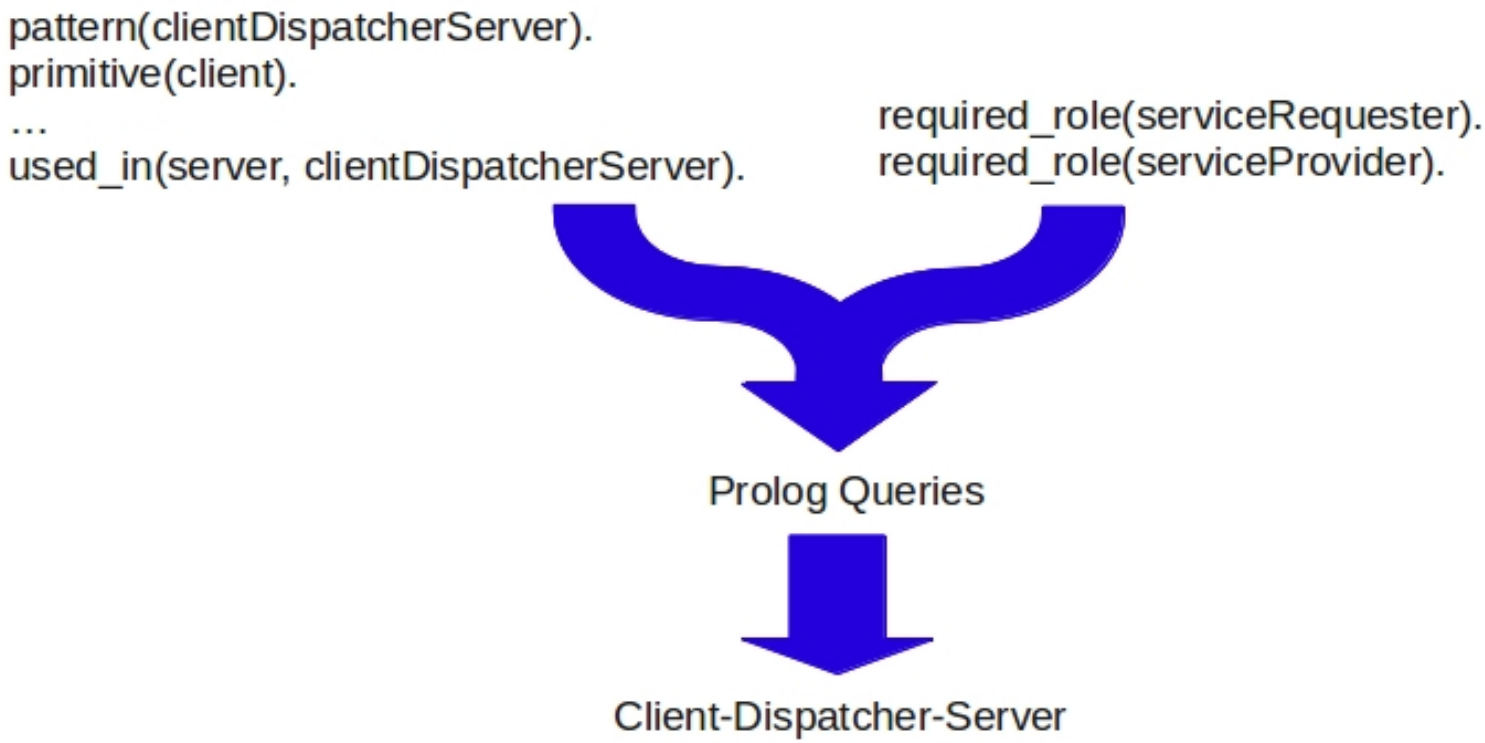
- *Graphicalization* process transforming the logical representation into an entity-relationship diagram
- Intuitively close to the representation of software models and patterns

Step 3: Select applicable patterns leveraging roles

Method:

- A pattern is applicable if it contains a fulfilling primitive for each assigned role
- Select patterns by combining the knowledge from the two previous steps

Implementation:



Conclusion

We proposed a *novel automated approach to selecting patterns* based on the observation that both *software models* and *patterns* are *relational*

Advantages

- ✔ Intuitive representation of both software models and patterns
- ✔ Significant amount of time saved during software development
- ✔ Provide hints on how to instantiate selected software patterns

Future work

Learning roles is difficult and requires further research

Even a semi-automatic role assignment saves a lot of time

Towards an Automated Pattern Selection Procedure in Software Models

Alexander van den Berghe, Jan Van Haaren,
Stefan Van Baelen, Yolande Berbers and Wouter Joosen

{firstname.lastname}@cs.kuleuven.be
IBBT-DistriNet, Department of Computer Science, KU Leuven
Celestijnenlaan 200A, 3001 Leuven, Belgium

22nd International Conference on Inductive Logic Programming
Monday September 17th 2012

KATHOLIEKE UNIVERSITEIT LEUVEN DistriNet RESEARCH GROUP

ibbt Security Department
Security, Privacy & Trust of E-*