Discovering Coordination Patterns in Legacy Software

Universidade do Minho
Departamento de Informática

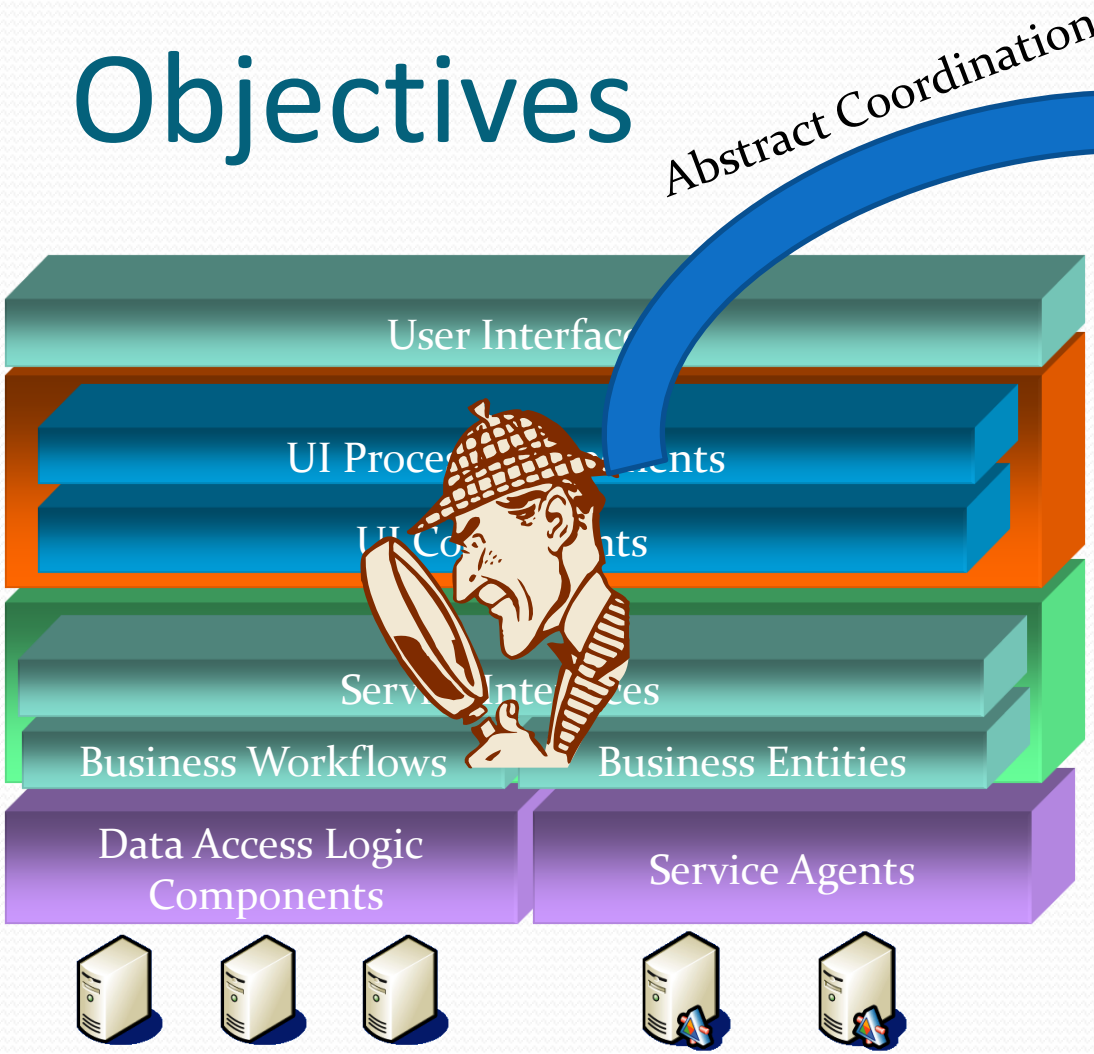
CIC'07 – Amsterdam– 23 Oct 2007

Nuno F. Rodrigues – nfr@di.uminho.pt
Luís S. Barbosa – lsb@di.uminho.pt

Agenda

- Objectives
- What are we looking for?
- Where can we look it for?
- How can we look it for?
- What else can we look for?

Objectives



ORC

```
P(d, x) :=
    (CNN(d)|BBC(d))
     > x > email(a,x)
```

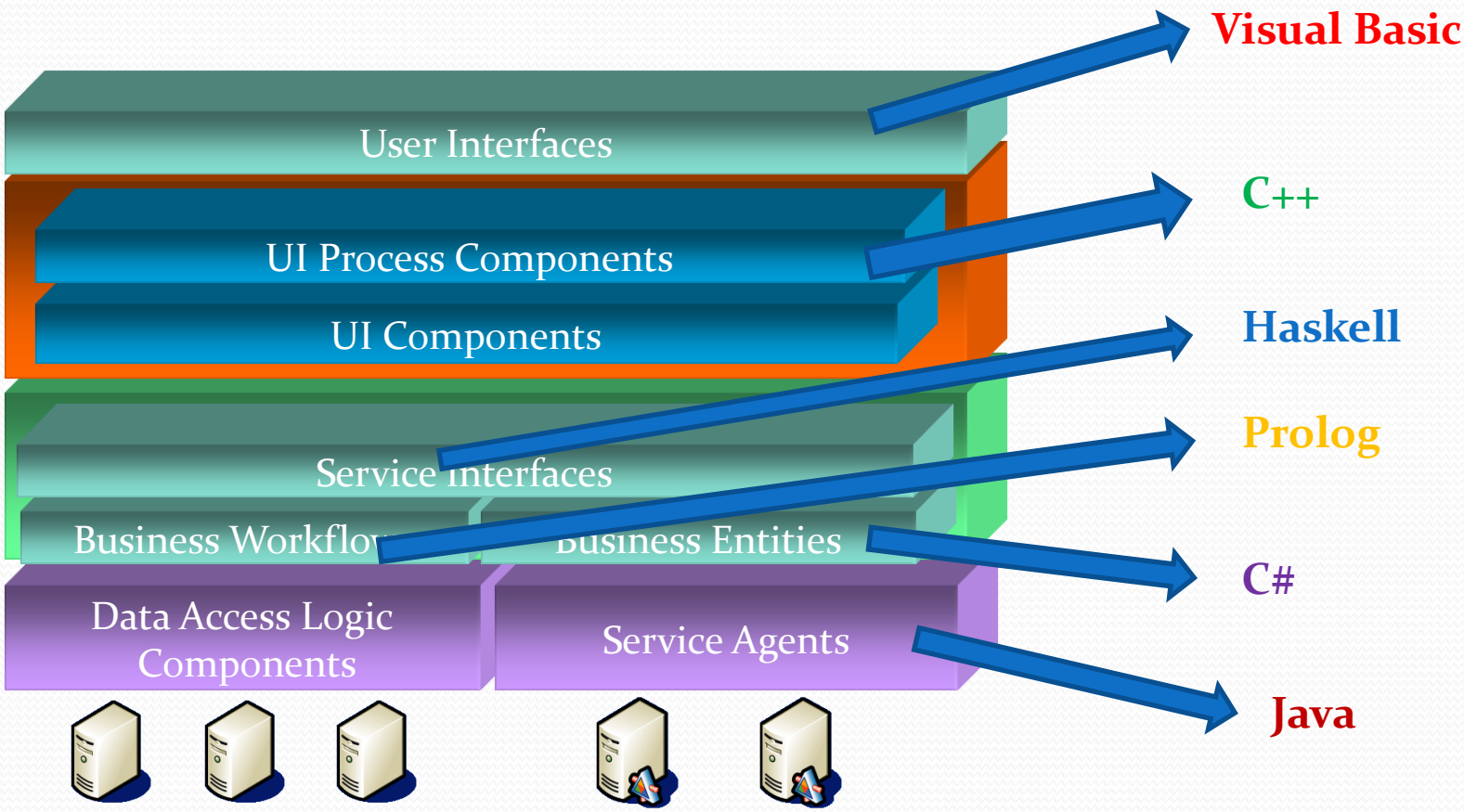
- Not just Web Services
- Thread Communication
- Remoting
- COM and CORBA
- etc

What are we looking for?

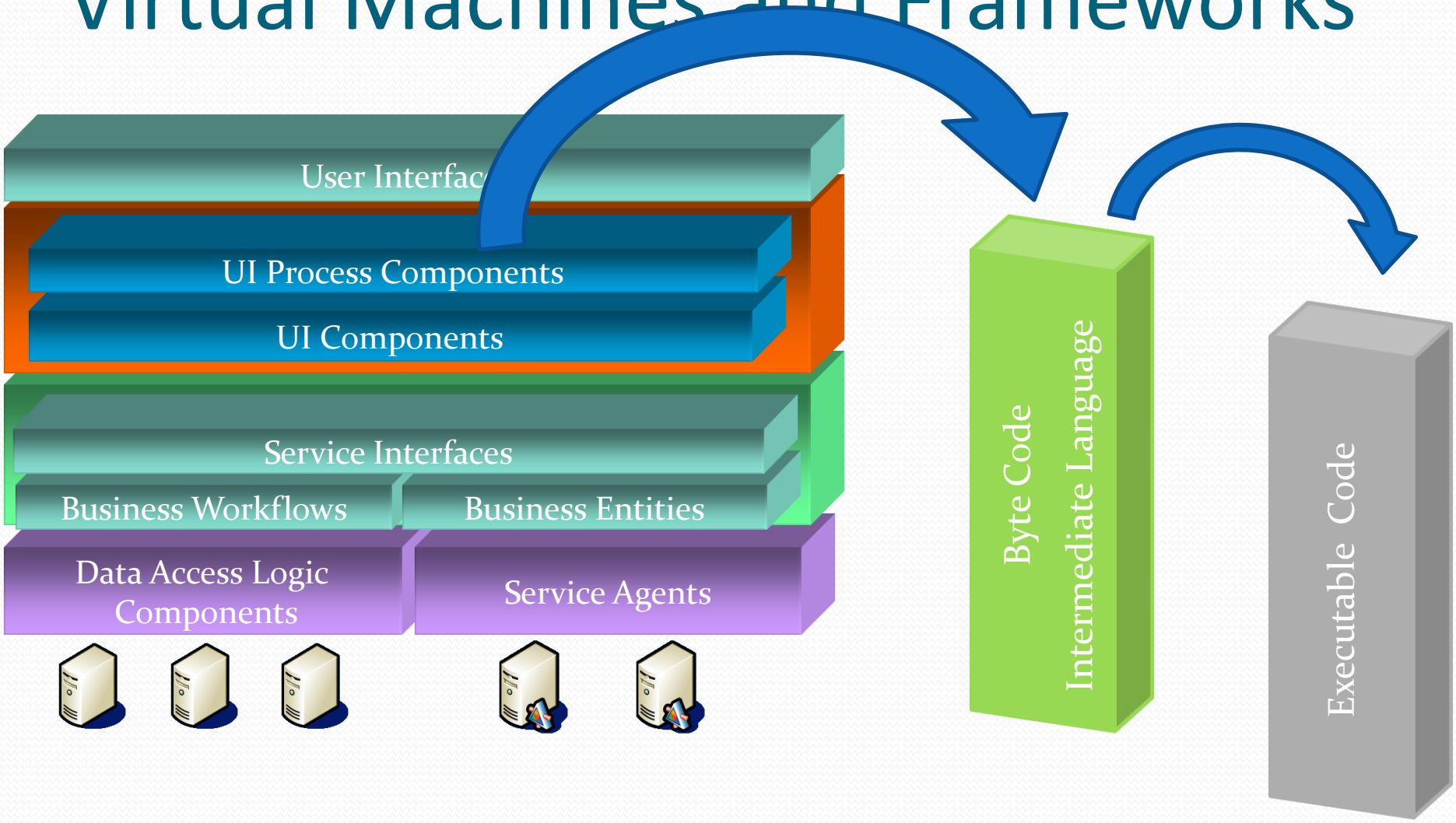
- Coordination Patterns using Web Services calls
  - Synchronous
  - Asynchronous
- But we also need to keep the code that regulates Web Service Calls

```
flights findFlight(string whereAmI) {
        if(whereAmI.equals("Amsterdam"))
                return KlmReservationWebService();
        else
                return TapReservationWebService();
}
```

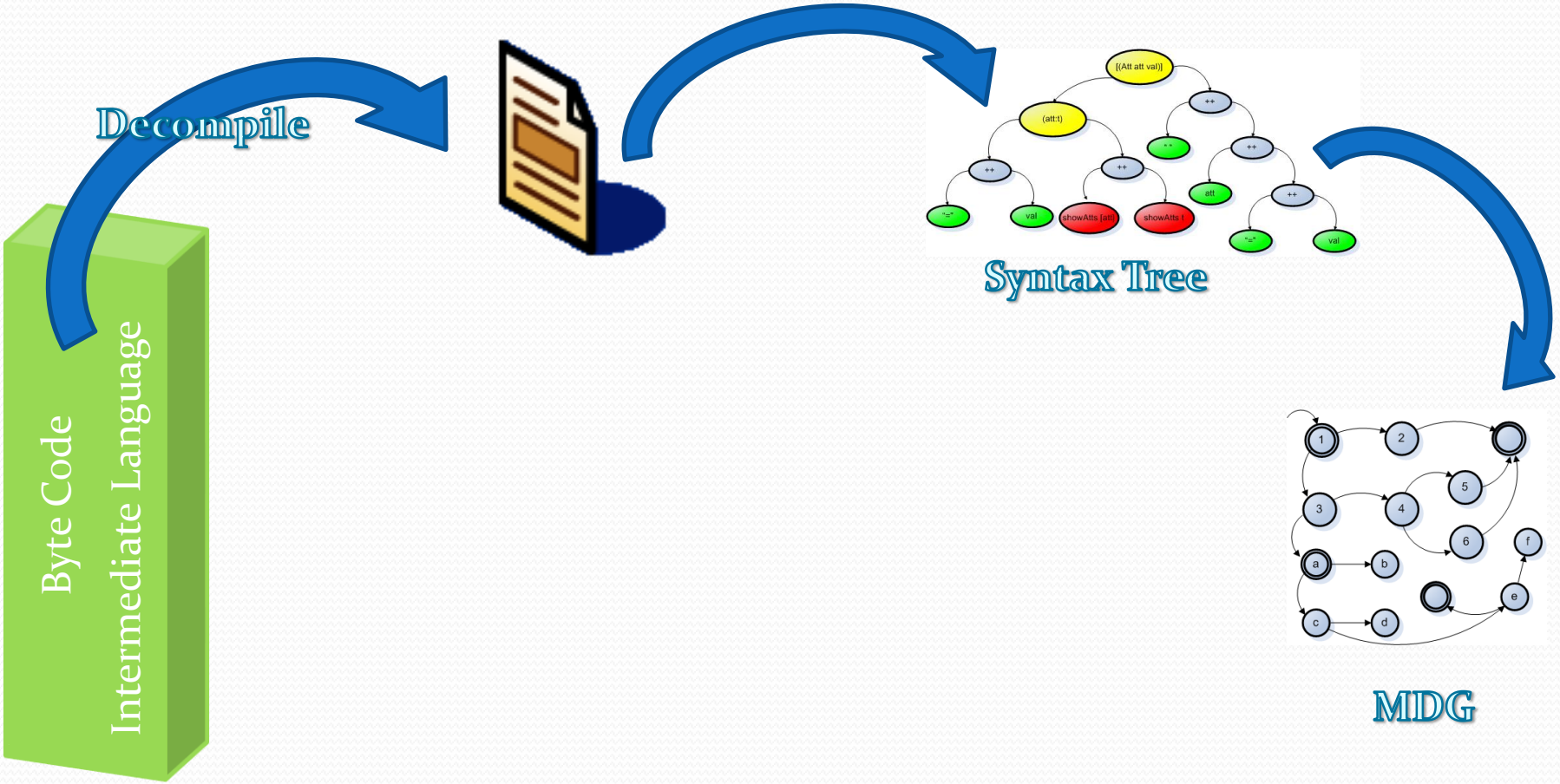
Where to look it for?



Virtual Machines and Frameworks



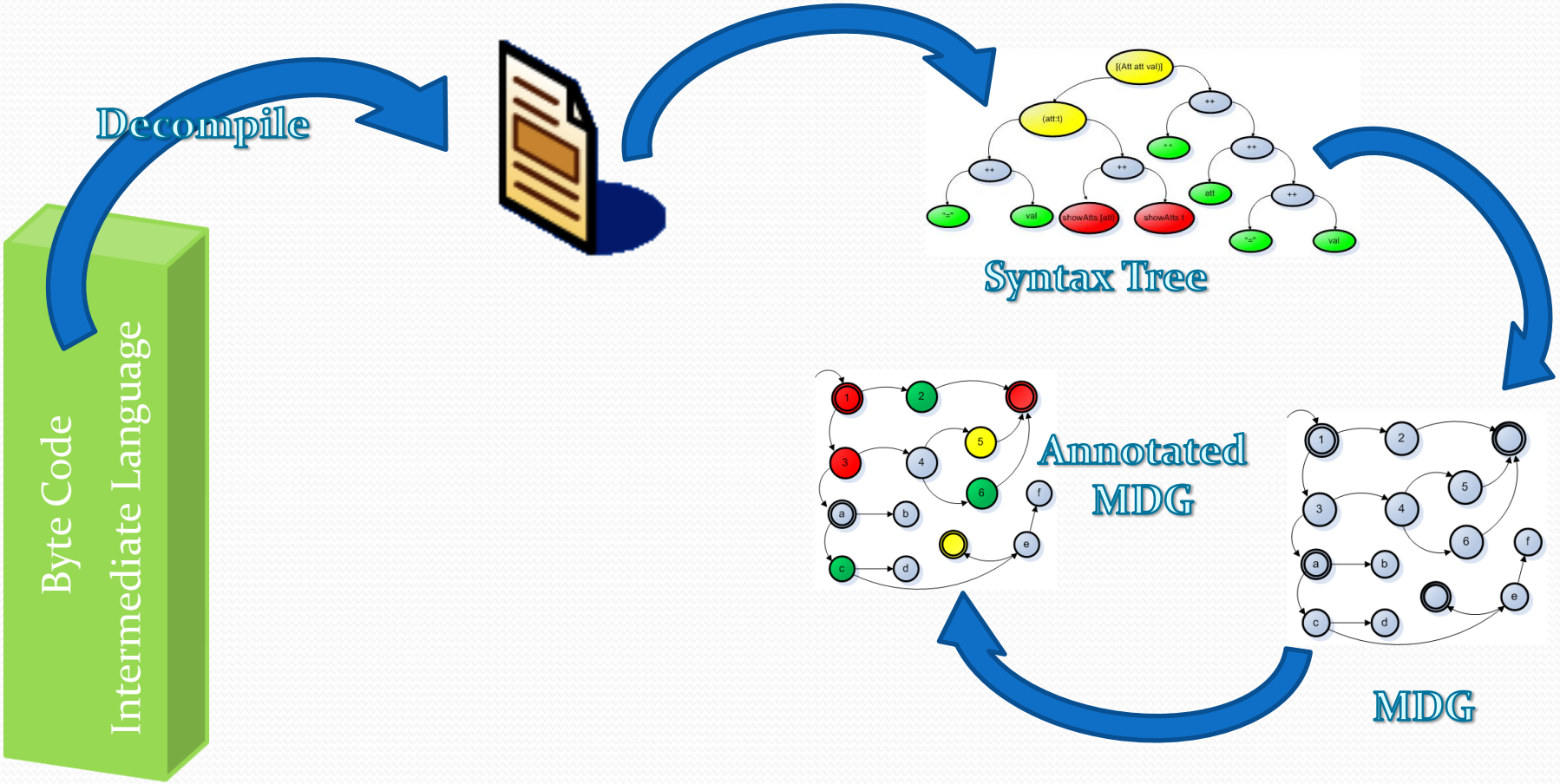
How can we look it for?



Syntax Tree -> MDG

- Create a node for each statement
- Create flow edges between control flow dependent statements
- Create data dependency edges for data dependent statements (*def* and *use* variables)
- If statement contains function call, then create MDG for the called function and connect it to the analyzed statement, creating new nodes for the actual parameters and return values

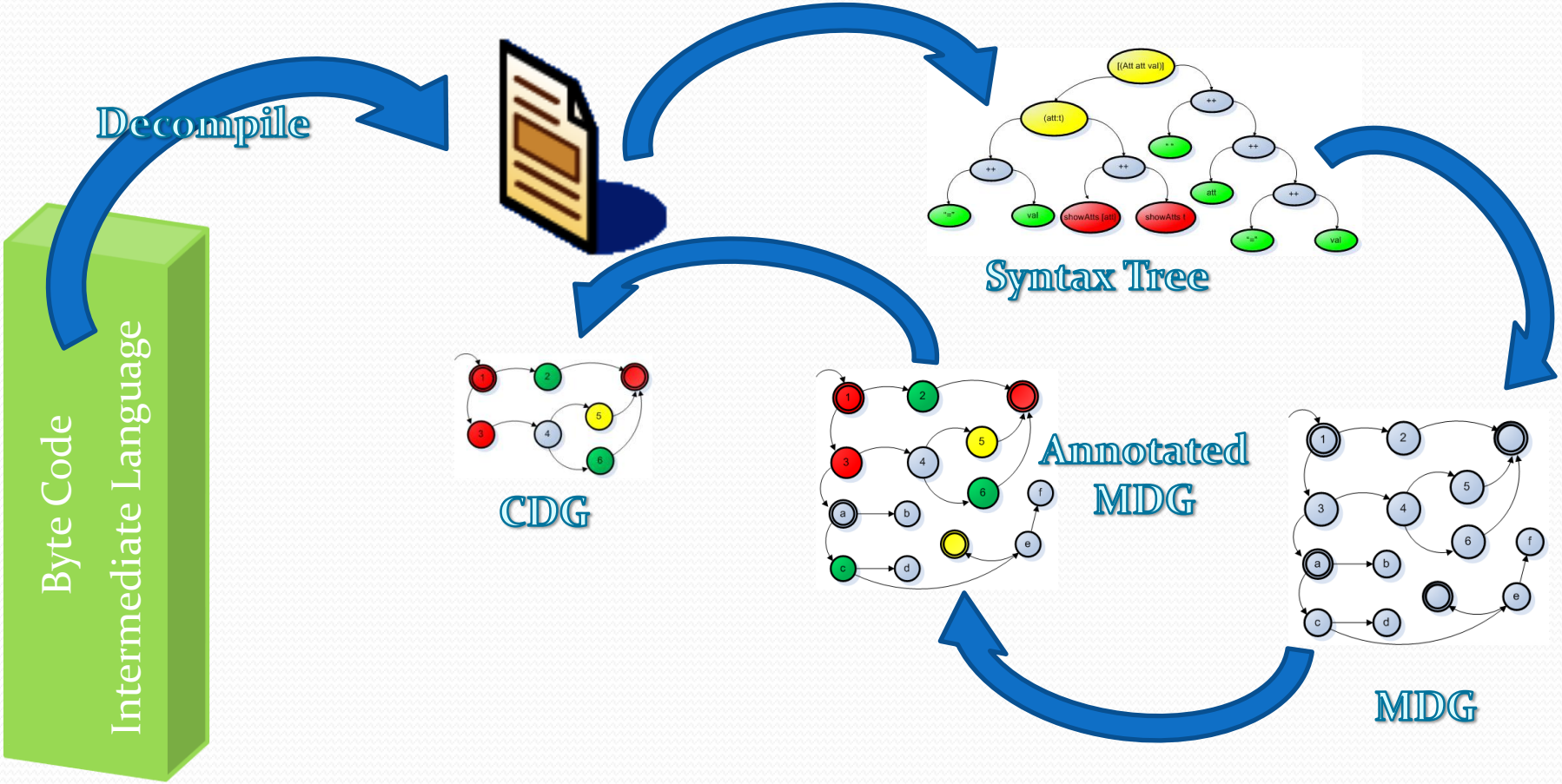
How can we look it for?



MDG -> Annotated MDG

- Parametric Annotations
  - Annotate Direct Web Service Calls (SOAP)
    - Synchronous
    - Asynchronous
  - Annotate COM and CORBA object calls
    - Synchronous
    - Asynchronous
  - Annotate Object Remoting calls (RMI, .net Remoting)
    - Synchronous
    - Asynchronous
  - Annotate Inter Thread Calls
    - Synchronous
    - Asynchronous
  - RSS, REST, …

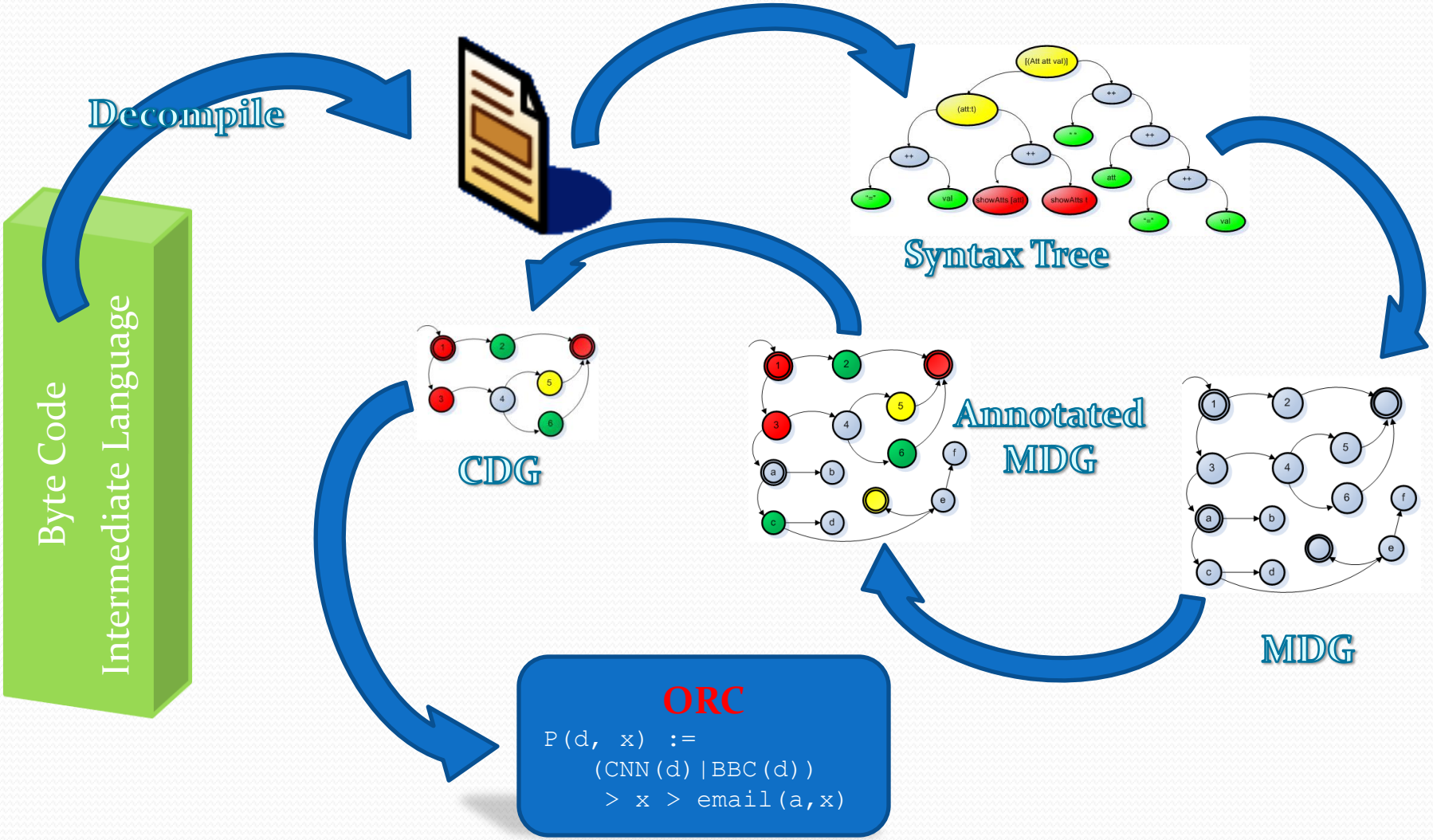
How can we look it for?



Annotated MDG -> CDG

- Remove all nodes that are not annotated except:
  - 1) Function call nodes for which there is a path to an annotated node
  - 2) Control flow nodes for which there is a path to an annotated node
  - 3) Keep the nodes in the backward slice of 2)

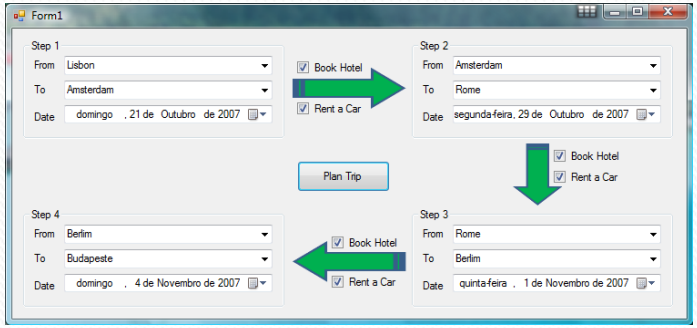
How can we look it for?

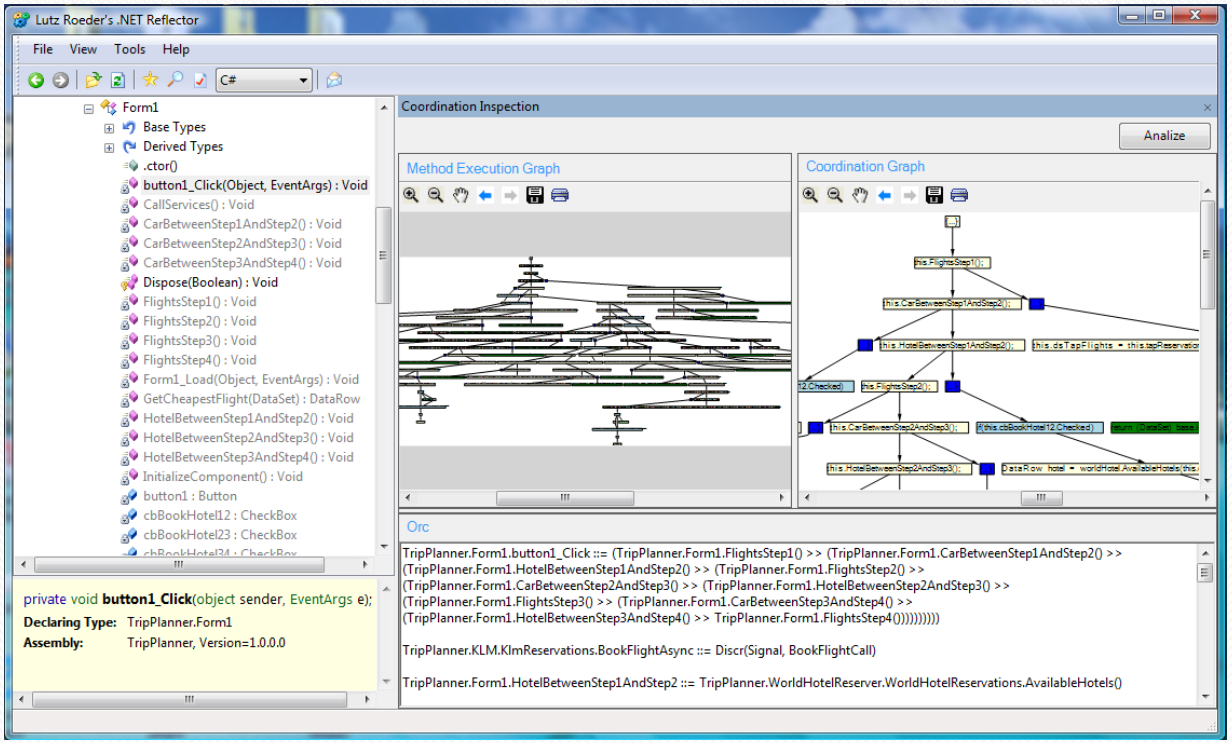


CDG -> Orc

- Traverse the CDG and
  - Generate Site Calls for the
    - Web Service calls
    - COM and CORBA calls
    - Remoting Calls
    - Threading calls
  - Generate Orc for the Control Flow nodes
    - XOR(a, b, c)
    - IF(a, b)
    - Recursive calls for While, For and Foreach
  - Simulate (in a different way) the internal workflow
  - Detect Recursive calls and generate equivalent Orc recursive definitions
    - Treat mutual recursion cases

Orc Inspector





Conclusion

- No need for source code
- Multi-language platform (>40 languages)
  - ADA, COBOL, Visual Basic, C#, J#, Delphi, C++, Prolog, Chrome, Ocaml, Mercury, JScript, PowerShell, Haskell, Python, Ruby, …
- Can be used on running applications
  - Useful for analyzing servers and other non-terminating systems
- The analysis method can cope with “Reflective Systems”
  - Since the analysis is performed on running systems, system reflection modifications are also detected

What else can we look for?

- First, implement all the light grey parts
- Look for more complex coordination patterns
  - p.e. Van der Aalst workflow patterns
  - At the Orc and Graph levels
- Generate “dummy” Orc for the called sites, so that the entire model can be animated
- If available, generate the actual Orc of the called sites
- Implement Orc expression transformation rules
- Implement properties verification over the generated Orc
- Generate skeleton implementations based on the transformed Orc
- If possible, generate running implementations based on the Orc generated and on the intermediate graphs

Thank you

Universidade do Minho

Departamento de Informática

PURe Project

Nuno Rodrigues – nfr@di.uminho.pt

Luís S. Barbosa – lsb@di.uminho.pt