Slicing of Component Behavior Specification w.r.t. their Composition

Ondřej Šerý and František Plášil

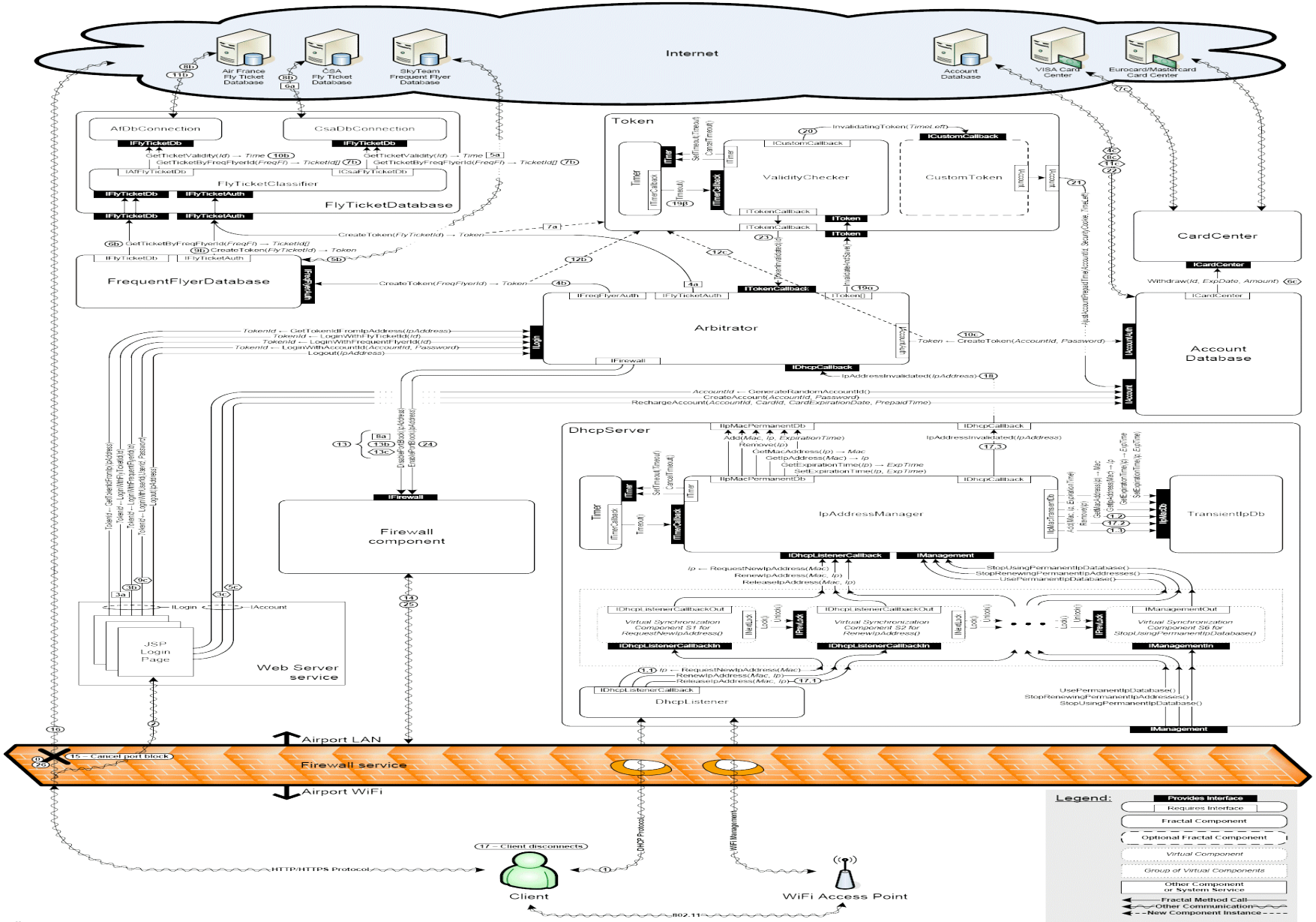
DISTRIBUTED SYSTEMS RESEARCH GROUP
http://dsrg.mff.cuni.cz

CHARLES UNIVERSITY PRAGUE
Faculty of Mathematics and Physics

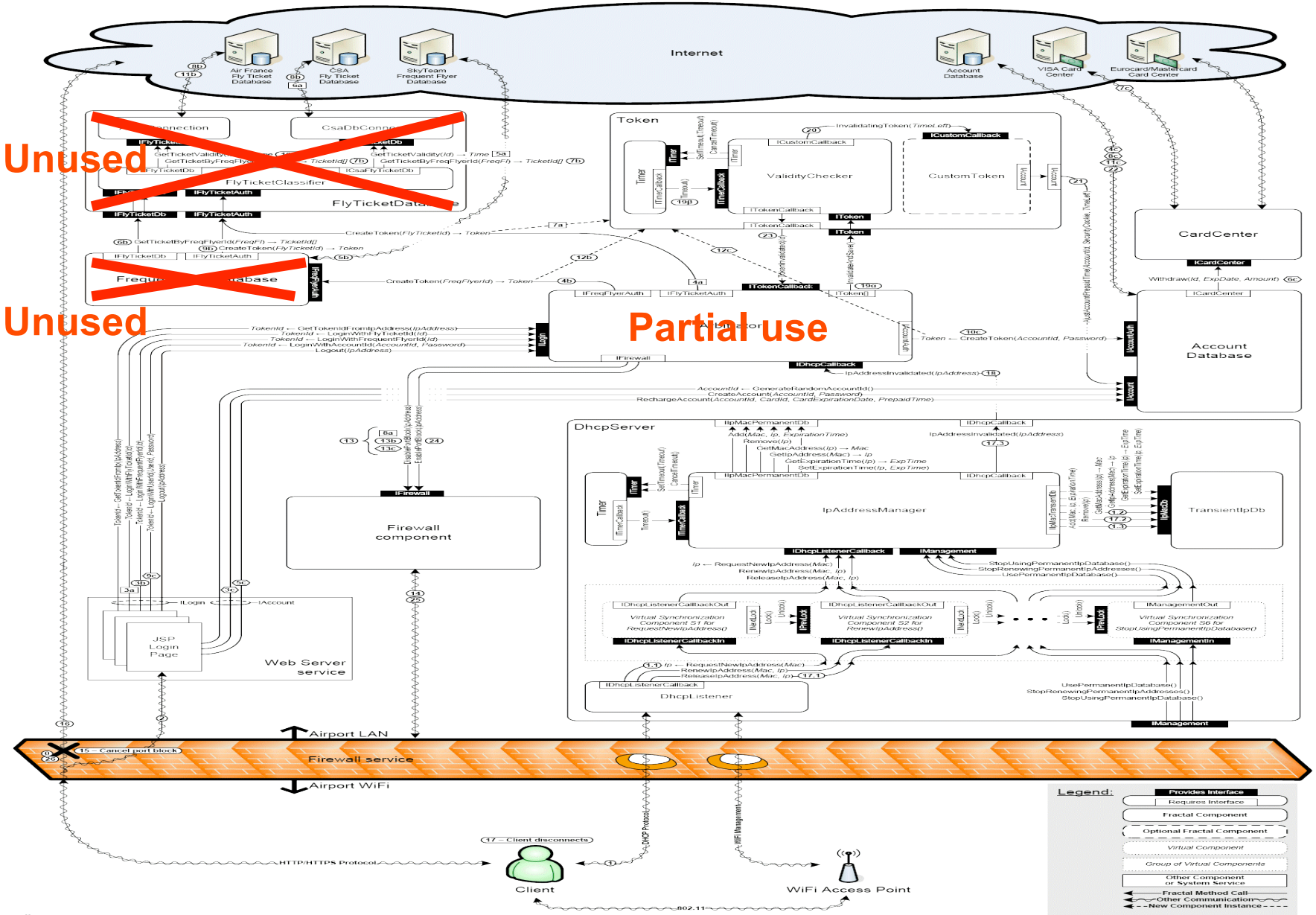
Goal

- Motivation
  - Observation: "*Reusable SW components usually provide more functionality than actually used in a concrete architecture/assembly*"
    - Behavior specification of such components is overspecified
    - Need for automatic slicing of the unused behavior, to
      - reveal actual roles of the components
      - make understanding of spec easier
- Goal
  - Slicing of behavior spec with respect to composition

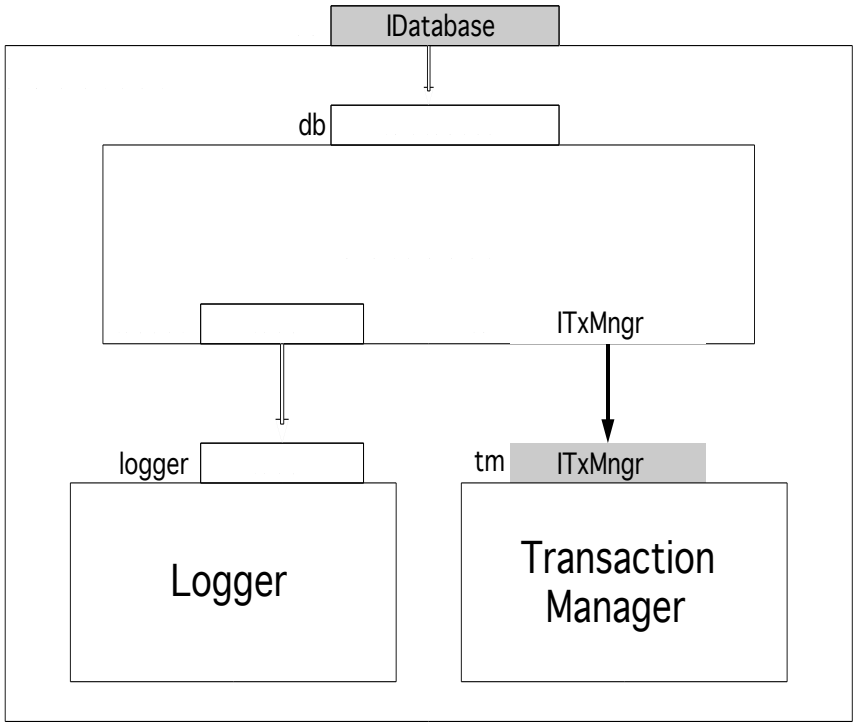
Motivation example 1: An airport internet providing application



Motivation example 2: Reuse in a public garden…



Database frame protocol




```
?db.start{!logger.start ; !tm.init} ;
(
  ?db.add{!tm.begin ; (!tm.commit +
                       !tm.rollback)}
  ||
  ?db.get{!tm.begin ; (!tm.commit +
                       !tm.rollback)}
  ||
  ?db.remove{!tm.begin ; (!tm.commit +
!tm.rollback)}
)* ;
?db.stop{!logger.stop ; !tm.destroy}
```

Behavior Protocols – syntax

- Behavior protocol
  - Expression describing the behavior of a software component
    - Infinite set of finite event traces

- Events:
  - Emitting a method call request: `!interface.method↑`
  - Accepting a method call request: `?interface.method↑`
  - Emitting a method call response: `!interface.method↓`
  - Accepting a method call response: `?interface.method↓`

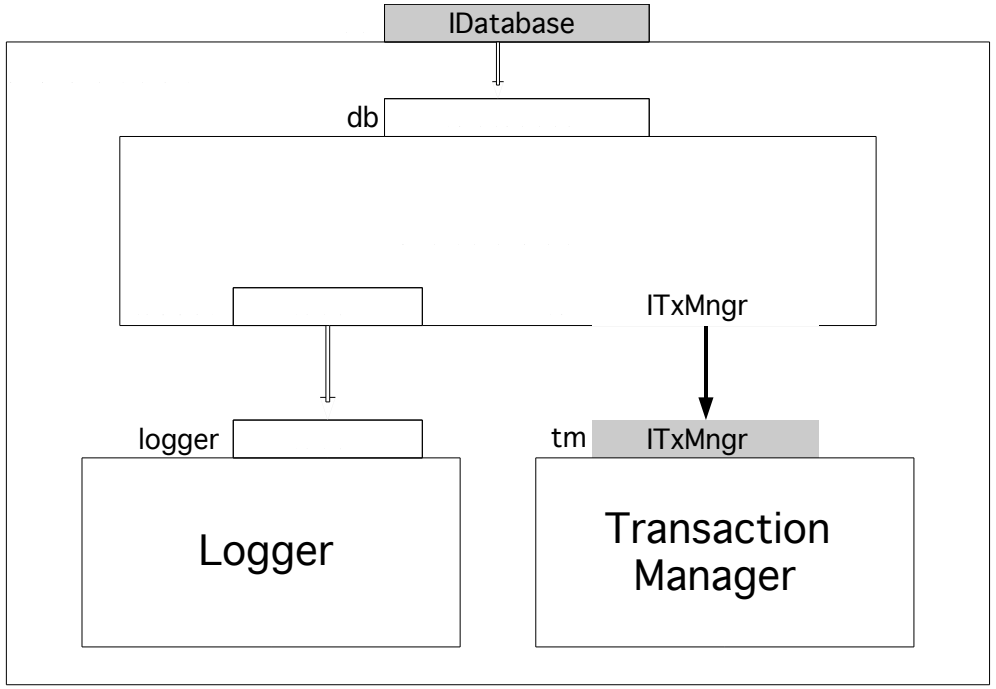
- Operators:
  - Sequence: **;**
  - Alternative: **+**
  - Repetition: *
  - And-parallel interleaving **|**
  - Or-parallel interleaving: **||**
  - **Consent** **∇**

    = parallel composition (interleaving + τ)
    indicating communication errors
    no activity (deadlock)
    bad activity (! cannot be responded)

- Syntactic abbreviations (to express method calls)
  - `?i.m` = `?i.m↑ ; !i.m↓`
  - `?i.m{`*prot*`}` = `?i.m↑ ;` *prot* `; !i.m↓`

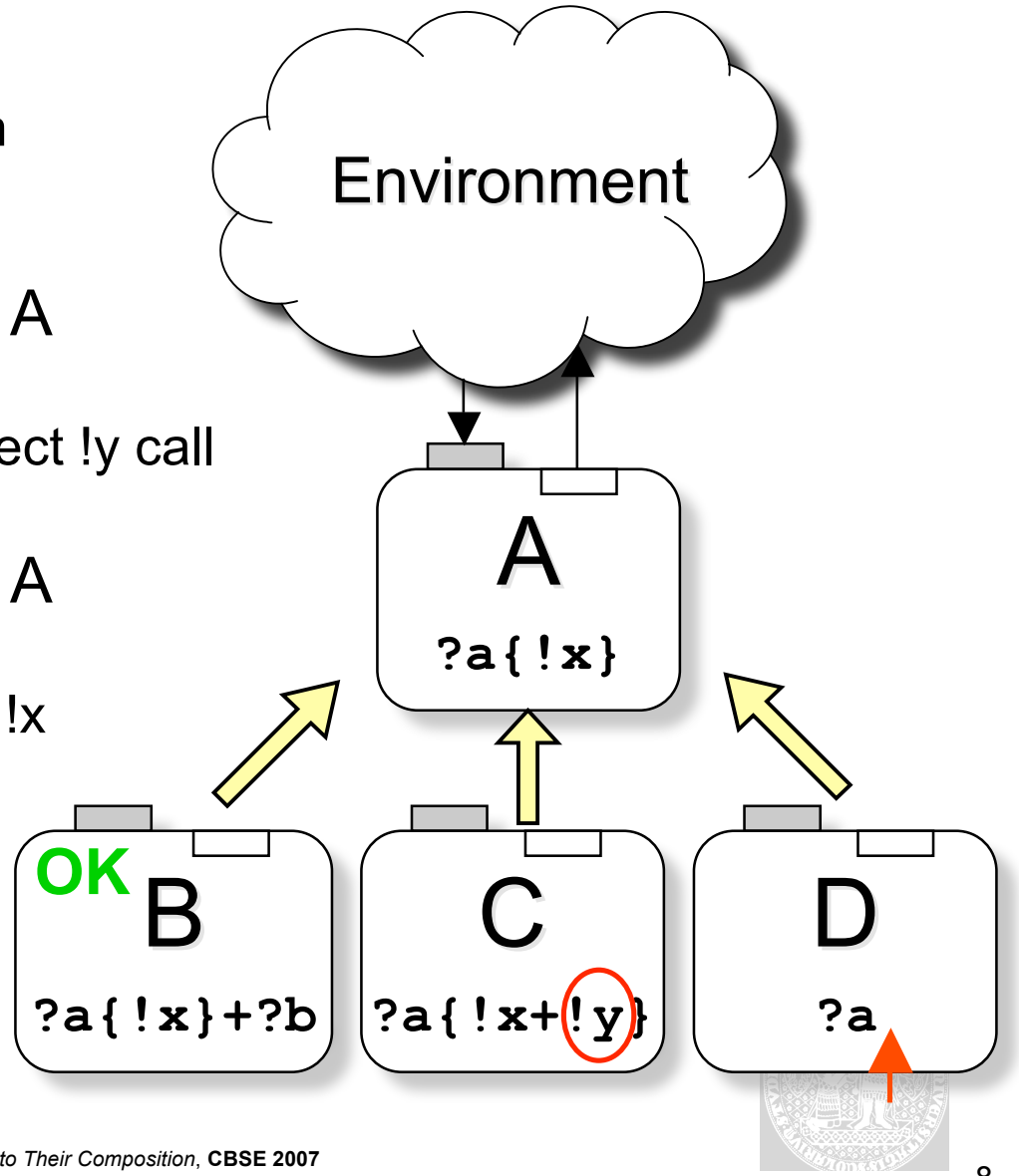
Behavior Compliance

- Horizontal compliance
  - $\mathtt{Database}_{FP} \nabla \mathtt{Logger}_{FP} \nabla \mathtt{Transactionmanager}_{FP} = \mathtt{Architecture\_prot}$
    - Tool: Behavior protocol checker (BPC) - Checks for communication errors
- Vertical compliance
  - $\mathtt{Architecture\_prot} \nabla \mathtt{DBServer}_{FP}^{-1}$
    - Tool: Behavior protocol checker (BPC) - Checks for communication errors



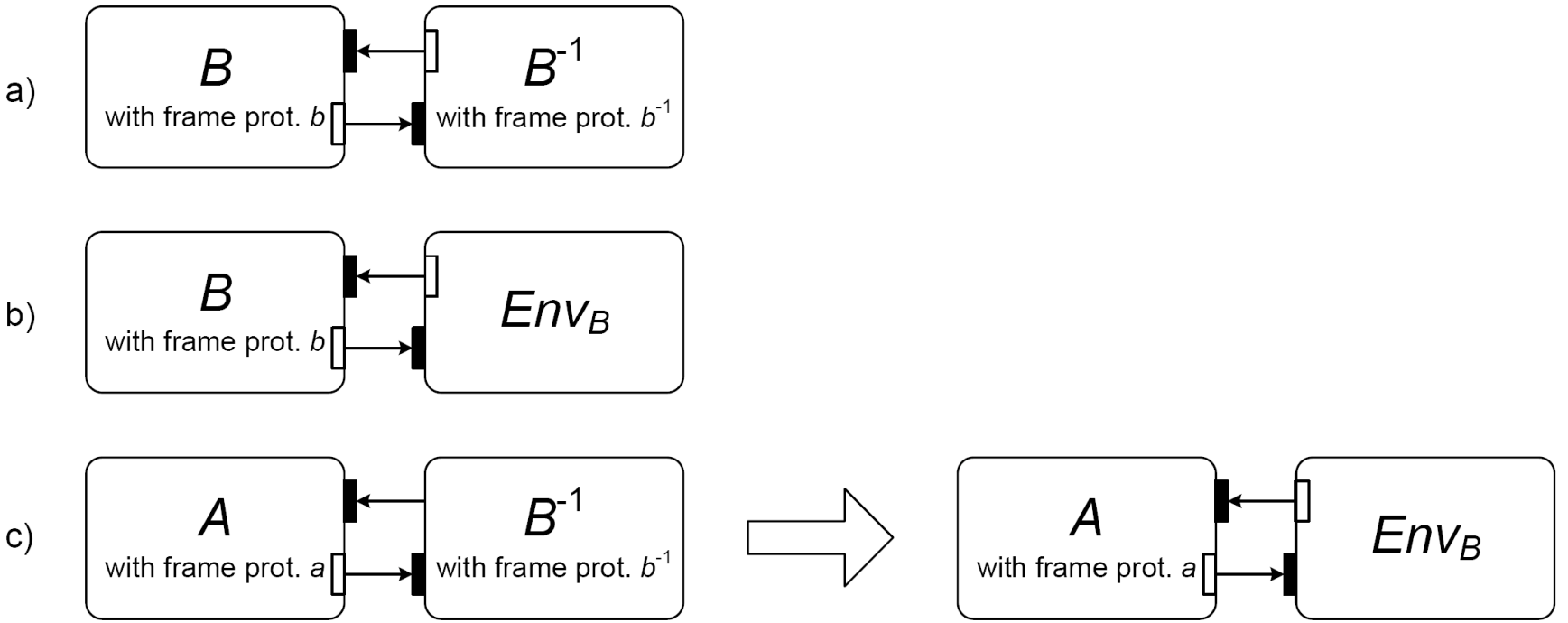
Substitutability – Example

- B is *substitutable* for A
  - ?b is not used in the given environment
- C is not *substitutable* for A
  - Bad-activity
  - Environment may not expect !y call
- D is not *substitutable* for A
  - No-activity (deadlock)
  - Environment may wait for !x



Substitutability

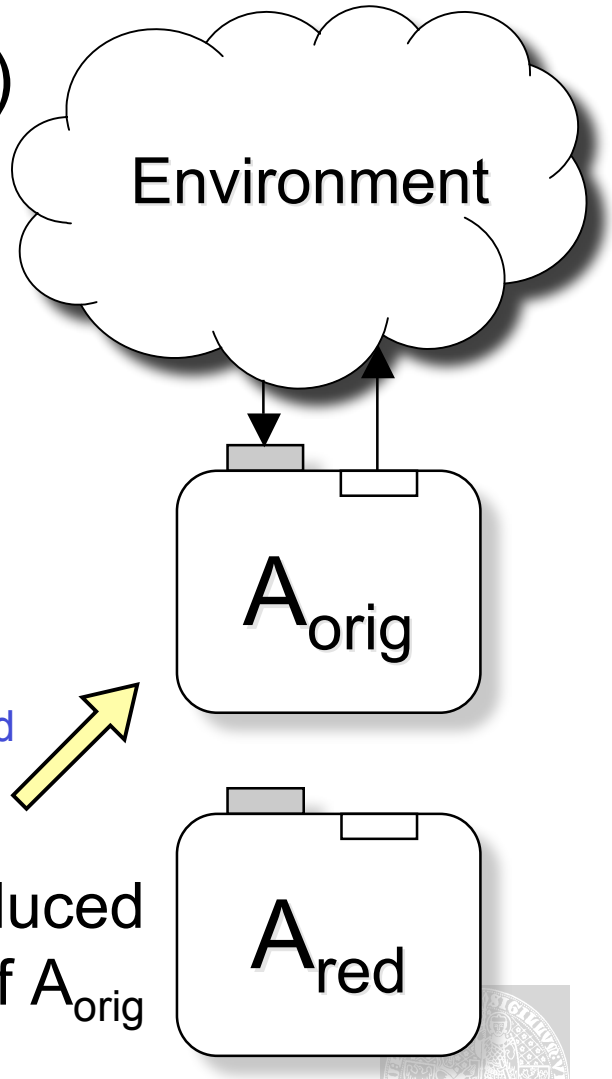
- **Def:** Protocol *a* is *substitutable* for *b* if $L(a \nabla b^{-1})$ does not contain any communication error
- B^{-1} is the most general environment of B



Reduction

- Motivation (A_{orig} is overspecified)
  - Replacing A_{orig} by A_{red}
    - A_{red} is a reduced variant of A_{orig}
  - Guarantee that:
    - Reasoning about $A_{red} \nabla$ Environment will also apply to A_{orig}

→ A_{orig} has to be substitutable for A_{red}



Reduction

- To put it more formally…

- **Def:** A protocol ***X*** is a ***reduction*** of ***Y*** if
  - ***Y*** is substitutable for ***X***
  - $L(\boldsymbol{X}) \subseteq L(\boldsymbol{Y})$

  - Intuitive extension to a *minimal reduction*

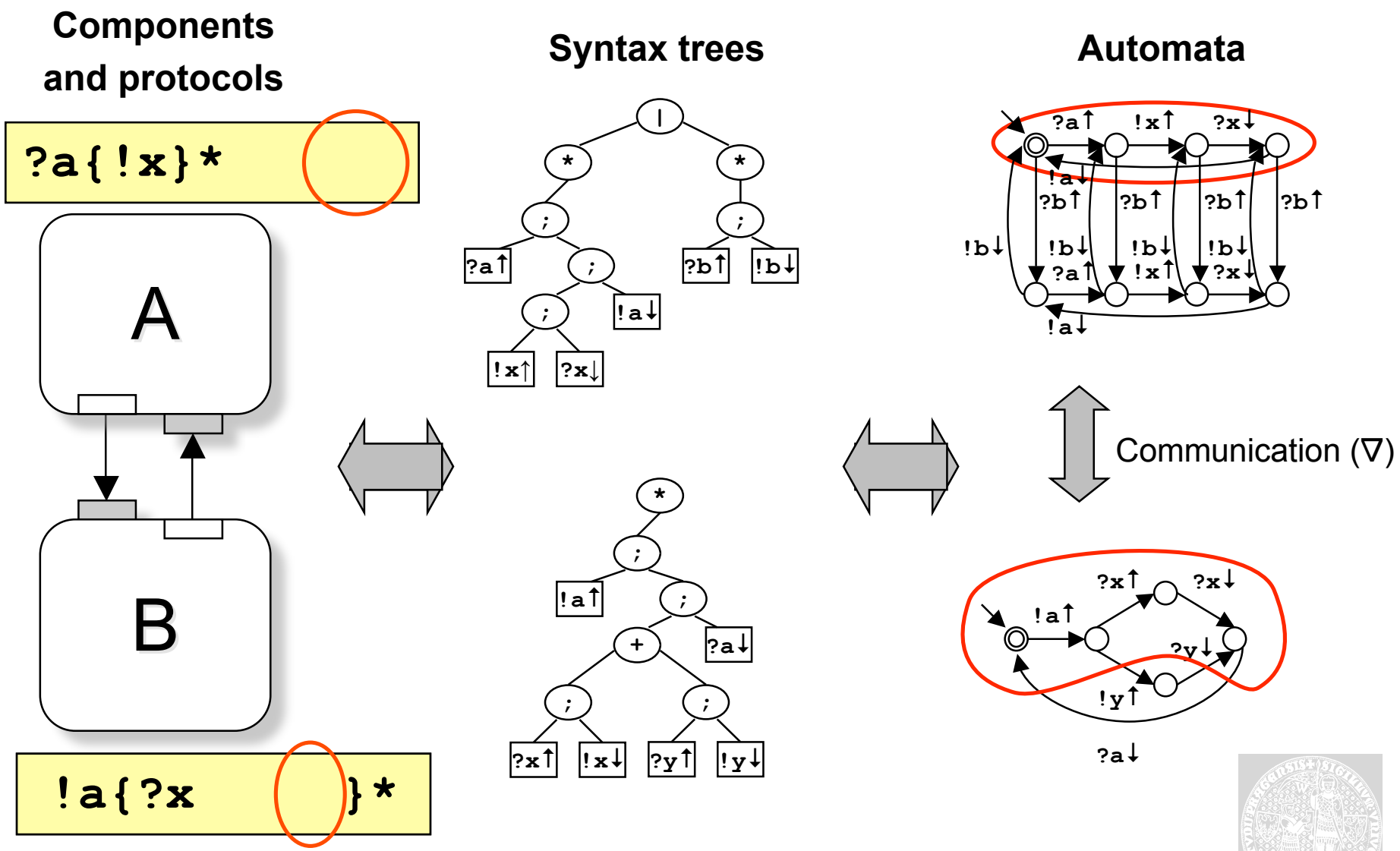
Slicing

- Why slicing (is *reduction* not enough)?
  - Reduction considers only semantics not syntax
    - Fewer behavior ≠ simpler protocols

→ *Slicing* – based on pruning the syntax trees
  - Resulting protocols are syntactically simplified

- **Def:** A protocol ***X*** is a ***slice*** of ***Y*** if
  - ***Y*** it is substitutable for ***X***
  - The syntax tree of ***X*** can be derived by pruning the syntax tree of ***Y***

Slicing w.r.t. composition



Implementation

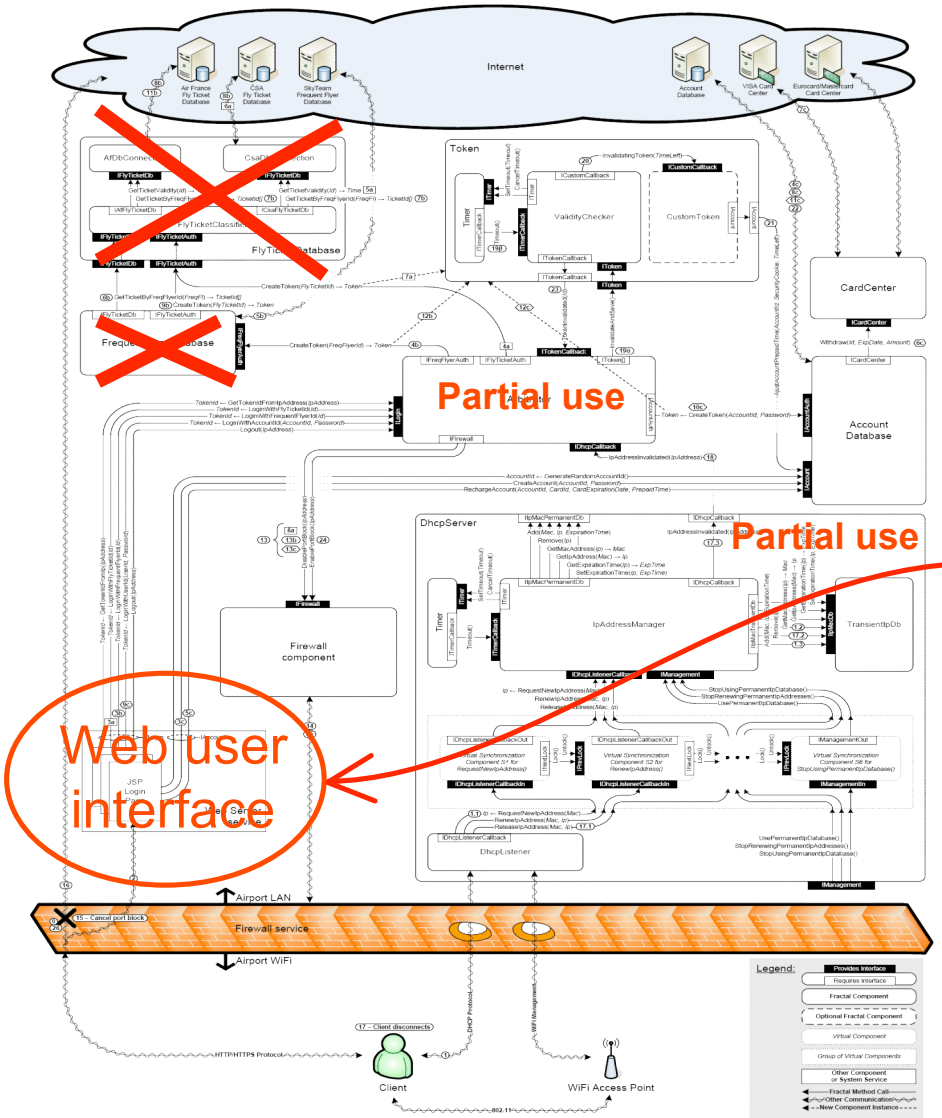
- Implemented as an extension to the behavior protocol checker – dChecker:
  - Available at:

    http://dsrg.mff.cuni.cz/projects/dchecker

- Tested on the demo application
  - Developed as a part of the Fractal Component Reliability Extension project:

    http://fractal.objectweb.org/fractalbpc

Case study



- Airport scenario
  - Frequent flyers
  - Business + 1st class passengers
  - Payment by a credit card
- Public garden scenario (reuse)
  - Only a credit card payment

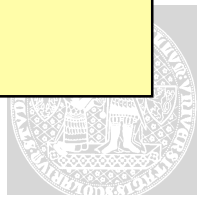
Protocol characterizing the environment:

```
(
    ?ILogin.GetTokenIdFromIpAddress +
    ?ILogin.LoginWithFlyTicketId +
    ?ILogin.LoginWithFrequentFlyerId +
    ?ILogin.LoginWithAccountId +
    ?ILogin.Logout +
    ?IAccount.GenerateRandomAccountId +
    ?IAccount.CreateAccount +
    ?IAccount.RechargeAccount
)*
```

Reducing protocol (1)

Arbitrator

```
(  :
   ?ILogin.GetTokenIdFromIpAddress +
   ?ILogin.LoginWithFlyTicketId {
        !IFlyTicketAuth.CreateToken_1 ;
        (!IFirewall.DisablePortBlock + NULL)} +
   ?ILogin.LoginWithFrequentFlyerId {
        !IFrequentFlyerAuth.CreateToken ;
        (!IFirewall.DisablePortBlock + NULL)} +
   ?ILogin.LoginWithAccountId {
        !IAccountAuth.CreateToken ;
        (!IFirewall.DisablePortBlock + NULL)} +
   ?ILogin.Logout {!IToken.InvalidateAndSave_1 + NULL}
)* |
?ITokenCallback.TokenInvalidated_1 {!IFirewall.EnablePortBlock_1}* |
?ITokenCallback.TokenInvalidated_2 {!IFirewall.EnablePortBlock_2}* |
?ITokenCallback.TokenInvalidated_3 {!IFirewall.EnablePortBlock_3}* |
?IDhcpCallback.IpAddressInvalidated {
   !IToken.InvalidateAndSave_2 + NULL}*
```



Reducing protocol (2)

DHCPServer:

```
!IDhcpCallback.IpAddressInvalidated*
|
(
   ?IManagement.UsePermanentIpDatabase^ ; (
         !IIpMacPermanentDb.GetIpAddress*
         |
         (
                  !IManagement.UsePermanentIpDatabase$ ;
                  ?IManagement.StopUsingPermanentIpDatabase^
         )
   ) ; !IManagement.StopUsingPermanentIpDatabase$
)*
```

Related work

- Program slicing (traditional)
  - Finding a “slice” – a minimal form of a program exhibiting the behavior of interest
  - Based on the control and data flow analysis
  - Debugging, Software maintenance, Model Checking (state space reduction), …
    - Weiser, M: **Program Slicing**. In Proceedings of *International Conference on Software Engineering*, pp. 439–449, 1981.
- General slicing
  - Extending the idea to a general algorithm working on the syntax-tree of an expression
    - Sloane, A. M.; Holdsworth, J: **Beyond Traditional Program Slicing**. Proceedings of the *1996 ACM SIGSOFT International Symposium on Software testing and analysis*, pp. 180–186.

Related work

- Requirement specification slicing
  - Applying slicing on the specification-level
    - Hassine, J; Dssouli, R.; Rilling, J.: **Applying Reduction Techniques to Software Functional Requirement Specifications**. *System Analysis and Modeling 2004*, pp. 138–153.
- Architectural slicing
  - Applying slicing on the architectural-level
    - Stanfford, J. A.; Wolf, A. L.: **Architecture-level Dependence Analysis in Support of Software Maintenance**. In Proceedings of the *Third International Workshop on Software Architecture*, pp. 129–132, 1998. ACM.
    - Zhao, J: **A Slicing-Based Approach to Extracting Reusable Software Architectures**. Proceedings of the *4th European Conference on Software Maintenance and Reengineering*, pp. 215–223, 2000.
- *And other…*

Conclusion

- We propose slicing of behavior specification
  - → Simple straightforward technique:
    slicing with respect to composition
- Prototype implementation
  - Extension of the behavior protocol checker
- Case study
  - Demonstrating the technique on the demo airport/garden application

Questions...?

Thank you for your attention

Any questions?

- Answers also at:
  http://dsrg.mff.cuni.cz