Modeling and Verifying Behavior of Software Components in SOFA 2

Jan Kofroň

Charles University in Prague
Faculty of Mathematics and Physics

Czech Republic

Outline

- Behavior Protocols
  - Introduction
  - Experience
    - Modeling and model checking
  - Demo
  - Flaws
- Behavior Protocols Extensions
  - Description
  - Demo
  - Experience
- Conclusion and Future work

Need For Semantic Specification

- When combining components from various vendors
  - Compatibility is to be assured
    - Syntactic check (interface signatures) is not enough
    - Need for semantic specification

- Semantic specification
  - Captures important aspects of the behavior
  - Forms an abstraction of the components
    - Omits details
  - Provides information for developer
  - Examples: LTS, use cases, sequence diagrams, process algebra expressions, etc.

Behavior Protocols I.

- A way to describe component behavior
- Processes defined via expressions (like process algebra)
- Methods calls and responses:

```
( ?open;
       ( ?read  { !impl_read } +
         ?write { !impl_write }
       )*;
?close )*
```

- Behavior = the language generated by BP ~ possibly infinite set of finite traces
- Checking protocol compliance ~ component behavior compatibility

Behavior Protocols II.

- 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 **|**
  - **Consent** ∇

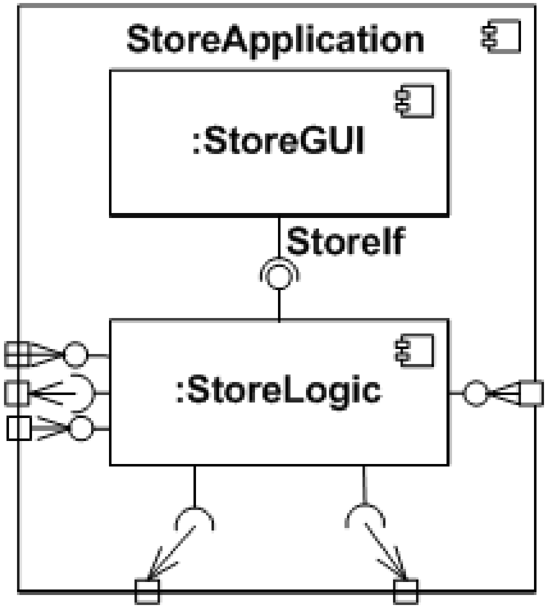
    = parallel composition (interleaving + τ)

    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
  - $\texttt{StoreGUI}_{FP} \nabla \texttt{StoreLogic}_{FP} = \texttt{Architecture\_prot}$
- Vertical compliance
  - $\texttt{Architecture\_prot} \nabla \texttt{StoreApplication}_{FP}^{-1}$
- Compliance
  - Absence of communication errors
    - Bad activity, No activity, Infinite activity
    - Can be found automatically
  - Verified separately for particular levels of nesting

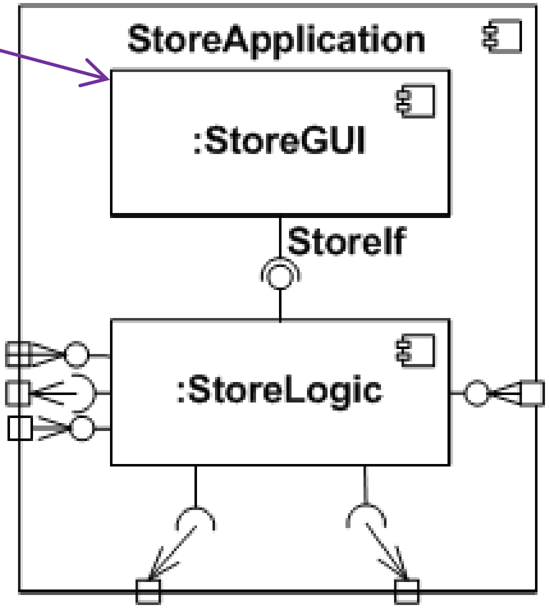


Verification of Behavior Compliance

- Behavior Protocol Checker (BPC)
  - Proprietary explicit state model checker for BP
  - Written in Java
  - Uses *Parse Tree automata* for state space generation
  - Able to verify state spaces of the order 10^7 states
    - May run several days
- dChecker
  - Again proprietary tool
  - Distributed state space traversal
  - Significantly faster than BPC
  - State spaces of the order 10^7 for each computer
    - i.e., entire state space of the order of 10^8 states

Demo

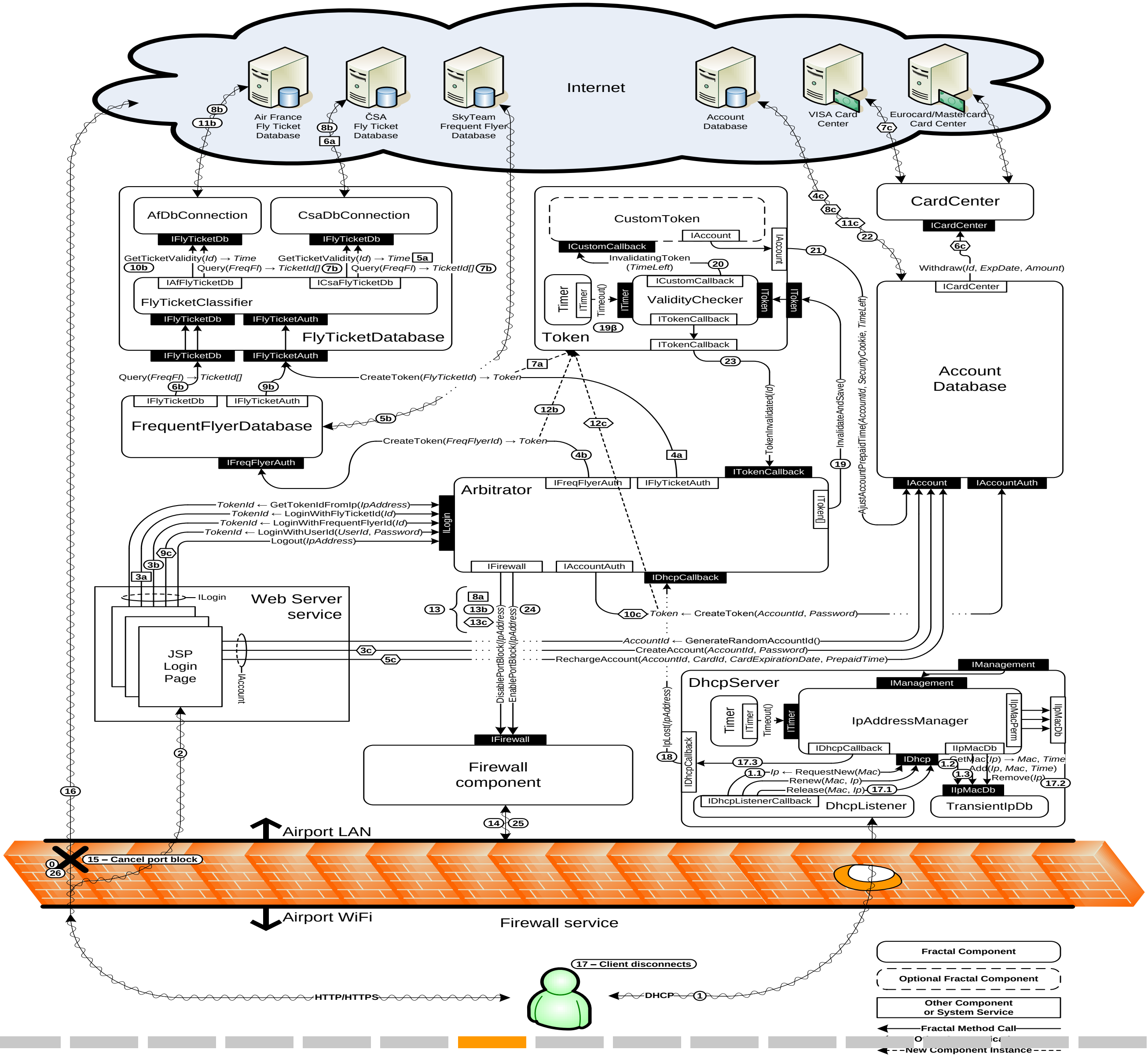
```
!StoreIf.getStore;
((
      !StoreIf.getAllProducts;
      !StoreIf.getProductsWithLowStock;
      !StoreIf.getAllProductsWithOptionalStockItem;
      !StoreIf.orderProducts*
    )
    +
    (
      !StoreIf.getOrder;
      !StoreIf.rollInReceivedOrder
    )
    +
    (
      !StoreIf.getAllProducts;
       !StoreIf.changePrice
))*
```



Evaluation of BP I.

- Case study for France Telecom
  - Fractal component application modeling the system granting access to Internet at airports
  - About 20 (primitive and composite) components
  - Verification of component compatibility (model level)
    - Simplification was necessary
    - Slightly simplified version took several hours
      - Simplification was necessary due to high (several GB) memory requirements
    - Models limited to size of 10^7 states – not enough in some cases
  - Obeying of model by implementation
    - Details – next talk by Pavel Parízek

CREF Demo Application



Evaluation of BP II.

- Several flaws of BP identified during the specification
  - Lack of synchronization mechanisms
    - Impossible to synchronize more than two components
  - Lack of expressiveness
    - Absence of macros caused parts repetition in the specification – hard to fix the errors
    - Absence of variables caused overspecification
    - Absence of a way to express common patterns, e.g. *until loops*, caused unreadable specification

Solution – Extended Behavior Protocols

- Extensions of BP
  - Data
    - Method parameters
    - Local variables
  - Synchronization
    - Special events – joining events – for synchronization of more than two components
  - Until loops
    - Just a syntactic abbreviation to enhance the readability

- Performance issues
  - Transformation of EBP into Promela – input language of Spin

Extended Behavior Protocols – Data

- Method parameters
  - Only of enumeration types
  - Because component behavior often depends on parameters
  - Not necessarily related to parameters in implementation

- Local variables
  - Again only of enumeration types
  - To model stateful components (e.g. component modes)
  - To store information across several method invocations

Extended Behavior Protocols – Synchronization

- Original BP allow synchronization of at most two components
  - Via ? - ! pair
  - Not enough in some cases – e.g. hierarchical initialization of components
- Special new kind of events – *multisynchronization events*

Extended Behavior Protocols – Until loops

- Just a syntactic abbreviation
  - Expressing until loops in original BP → extremely ugly protocol
  - Repetition controlled by value of local variable

EBP – Example

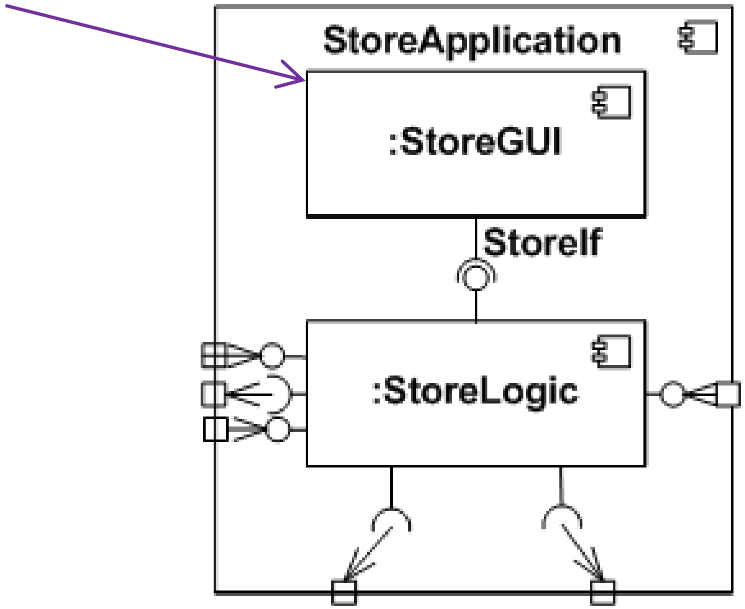
```
component LightDisplay {
   types {
      states = {LIGHT_ENABLED, LIGHT_DISABLED}
   }
   vars {
      states state = LIGHT_ENABLED
   }
   behavior {
      ?LightDisplayControllerEventHandlerIf.onEvent(ExpressModeEnabledEvent) {
         state <- LIGHT_ENABLED
      }*
      |
      ?LightDisplayControllerEventHandlerIf.onEvent(ExpressModeDisabledEvent) {
         state <- LIGHT_DISABLED
      }*
   }
}
```

Verification of Behavior Compliance II.

- Using ebp2promela tool
  - Translating EBP into Promela → Spin model checker
  - Faster, able to traverse larger state spaces
  - No need for maintaining a proprietary tool
    - Just the transformer – much easier

Demo 2

```
component StoreGUI {
  behavior {
    !StoreIf.getStore; #init
    (
      (
        !StoreIf.getAllProducts;
        !StoreIf.getProductsWithLowStock;
        !StoreIf.getAllProductsWithOptionalStockItem;
        !StoreIf.orderProducts*
      )     +     (
        !StoreIf.getOrder;
        !StoreIf.rollInReceivedOrder
      )     +     (
        !StoreIf.getAllProducts;
        !StoreIf.changePrice
      )
    )*
  }
}
```



Experience

- Both BP and EBP applied on the CoCoME application
  - Part of modeling contest
  - Behavior of entire application modeled and verified
  - EBP turned out to be better than BP
    - Easier to write/read/maintain, verification faster
    - More precise ~ less abstract
  - Abstract from
    - Ordinary data
    - Non-regular behavior
      - E.g. recursion
    - Threads
      - Parallelism modeled to some extent

Conclusion and Future Work

- Verification of EBP against implementation
  - Similar to BP-against-code verification
- Evaluation on more case studies
  - Further enhancements/modification to cover discovered flaws
- Generation of skeletons of components
  - Keeping important aspects of behavior spec

Thank you!