Datenstrommodellierung in Software-Architekturen: Konzepte, Analysen und offene Fragen

Prof. Dr. Ralf Reussner, Stephan Seifermann
GI Architekturen 2016
Quality as Key of Success
Quality Engineering

Quality attributes
- Performance
- Reliability
- Security
- Compliance

Quality by design
- Modeling and analyzing of design
- Determine quality before deployment
- Cost-efficient fixing of issues

Data flows as first class entities enable or enhance quality analyses
- Security (e.g. confidentiality)
- Compliance (e.g. handling of personal information)
- Performance (e.g. big data stream processing)
## State of Data Flow Modeling

### Threat modeling
- High level of abstraction
- Often solely done for threat analysis
- Security experts create and interpret models

[Swiderski and Snyder 2004]

### Implementation analyses
- Low level of abstraction
- Break down of high level to low level goals
- High precision results regarding information flows, inference, …

[Ahrendt et al. 2005]
[Snelting et al. 2014]

### Architectural data flows
- Inferred from control flow via parameters
- Separate models

[Kramer et al. 2014]
[Schmieders et al. 2015]

---

Data is no first class entity in architectural design phase
PIBA

Problem
- Data and their properties are highly relevant for certain quality analyses
- No annotation of architectural elements with data possible

Idea
- Introduce data and data flows as first class entities
- Use data properties in quality analyses

Benefit
- Directly adopt information from requirements engineering in architecture
- Express quality in natural way

Action
- Extend Palladio with data flow constructs
- Define quality analyses for common use cases using data flows
Fitting of Data Flows into Software Architecture

Architecture / Design
- Usage Description
- Structure
- Behavior
- Configuration Description
- Deployment Description

Implementation
- Source Code
- Usage
- Configuration
- Deployment

Runtime
- Control Flow
- Data Flow

- CF Perform. Abstraction realizes CF Reliability Abstraction
- DF Confident. Abstraction realizes DF Compliance Abstraction
- Control Flow defines Data Flow
- Usage represents Source Code
- Configuration realizes Deployment
- Behavior realizes Configuration Description
- Deployment represents Deployment Description
- Control Flow realizes Data Flow
- Usage represents Usage
- Configuration realizes Configuration Description
- Deployment represents Deployment Description

represents
- Control Flow
- Data Flow

enables
- Control Flow
- Data Flow

defines
- Control Flow
- Data Flow

realizes
- Control Flow
- Data Flow
ADL Modeling Extension

- Classes of data and relevant meta-data
- Data processors
- Data sources and sinks
- Data flow
  - Inside components: chained processors
  - Between components: transfer processors
The Palladio Approach

Palladio Component Model
- Quality analyses of component-based systems
- Reusable specification

Analyses
- Performance
- Reliability
- Maintainability / costs

Development Process and Tools
- Process for development of component-based software
- IDE for development and analyses

[Reussner et al. 2016]
Views on System Models in Palladio

- Holistic view on the system
  - Software and hardware
  - Static and dynamic views
  - Allocation and usage profile
Palladio Modeling Extension
Usage Profile

```
«SystemCallAction»
IFacade.register

«VariableBinding»
  parameter = userData
data = userBirthday

«VariableBinding»
  parameter = userData
data = userName

User

«DataDefinition»
Birthday
  id = userBirthday
  value = User.birthday

«DataDefinition»
Name
  id = userName
  value = User.name
```
```cpp
UserManagement::register(userData)

- «DataProcessing»
  - ReadData
    - data = userName
    - param = userData

- «DataProcessing»
  - ReadData
    - data = userBirthday
    - param = userData

- «DataProcessing»
  - EncryptData
    - data = userName
    - name = userNameEnc

- «ExternalCallAction»
  - DB.RegisterUser
    - params = {userData}

- «VariableBinding»
  - BindRegistrationData
    - data = registrationData
    - param = userData

- «DataProcessing»
  - TupelizeData
    - data = {userNameEnc, userBirthday}
    - name = registrationData
```
**Palladio Modeling Extension**

**Summary**

24.06.2016

Datenstrommodellierung in Software-Architekturen - Ralf Reussner

---

**Usage Profile**

**Software Components**

**Resource Environment**

---

**Palladio Extension**

**Information**

Start of data flow  
Data definitions  
Data bindings

**EntryLevelSystemCallIfacade.register()**

**«DataDefinition»**  
**Birthday**

id = userBirthday
value = User.birthday

**register()**

**«DataProcessing»**  
**ReadData**

**«ExternalCallAction»**  
**UserMgmt.register**

**«DataProcessing»**  
**EncryptData**

**Node A**

**Property location = USA**

**Node B**

**Properties of Containers/Links**

---

**«DataProcesing»**
Data Flow Analysis Approaches

<table>
<thead>
<tr>
<th></th>
<th>Analytical</th>
<th>Simulative</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input</strong></td>
<td>Extended ADL artifact, analysis specification</td>
<td></td>
</tr>
</tbody>
</table>
| **Method**           | • Transform inputs to constraints  
                        • use constraint solver |
|                      | • Transform ADL artifact to simulation code  
                        • use results to answer analysis |
|                      | • Translate results back to ADL artifact elements |
| **Areas of Applicability** | • Worst-case analyses |
|                      | • Concurrent behavior  
                        • State depending behavior |

**Diagram:**
- ADL Artifact → ADL with DFs Artifact
- ADL with DFs Artifact → Constraints / Simulation Code
- Goals/Preferences → Raw Results
- Raw Results → Transformed Results
Analyses Based on Data Flows

**Horizontal**
- Infer offered service quality for interface
- Determine fulfillment of user goals

**Vertical**
- Determine requirements on infrastructure

**Compliance**
- Location of storage or processing of personal data
- Approval before transmitting personal data

**Confidentiality**
- Influence of data on other data
- Confidentiality during transmission
Transformation into Constraint Logic Problem Example

**Resource Environment**
- Properties of resources: `resourceProperty(Server1, location, EU)`
- Linking between component and linking resource: `resourceLink(Server1, Network)`

**Allocation View**
- Mapping between resources and components: `allocated(Facade, Server1)`

Diagram:
- `Server1`:
  - CPU
  - Properties: `location = EU`
- `Server2`:
  - CPU
  - Properties: `location = USA`
- `Network`:
  - LAN
  - Properties: `encrypted = True`
Transformation into Constraint Logic Problem
Example

Usage Model
- Data Definition
dataDefinition(userBirthday)
- Variable Binding
bind(userBirthday, userData)

Usage Model / Component Specification
- Call actions
call(Facade, register, UserManagement, register, [userData])
- Data processor
applyEncryption(UserManagement, register, userName, userNameEnc)
- Ordering of actions
post(bind(...), call(...))
Transformation into Constraint Logic Problem

Example

Analysis goal

- No personal data d is transferred from the EU to other country.

\[ \text{allocated}(x, s \downarrow 1) \land \text{allocated}(y, s \downarrow 2) \land s \downarrow 1 \neq s \downarrow 2 \]
\[ \land \]
\[ \text{resourceProperty}(s \downarrow 1, \text{location}, l \downarrow 1) \land \text{resourceProperty}(s \downarrow 2, \text{location}, l \downarrow 2) \]
\[ \land \ l \downarrow 1 \neq l \downarrow 2 \]
\[ \land \]
\[ l \downarrow 1 \in \{\text{EU}\} \land l \downarrow 2 \notin \{\text{EU}\} \]
\[ \land \]
\[ \exists \text{bind}(d, p) \text{ with } p \in P \land \text{personal}(d) \]

- no call x to y with parameters P
- x, y deployed on different server
- servers in different locations
- x inside, y outside EU
- personal data is transferred via one of the parameters
Benefits of Data Flows as First Class Entities in ADLs

| Low modeling effort | • More natural expression of quality requirements  
|                      | • Reuse of existing specifications  
|                      | • Consistency given automatically  
| Comprehensible results | • Data flows defined in similar terms as architecture  
|                          | • Clear mapping from results to known elements  
| Usable by non-experts | • Stakeholders fokus on well known parts  
|                      | • No separate analysis models  
| Early issue detection | • No implementation required  
|                      | • Low cost what-if scenarios  
|                      | • Stable quality properties even after evolution  

24.06.2016 Datenstrommodellierung in Software-Architekturen - Ralf Reussner
Extensible Quality Analyses Based on Data Flows

- Basic Data Flow Modeling
- Extensible Annotations for Quality Modeling
- Analysis Infrastructure
- Data Properties
- Data Processors
- Quality Annotations
- Common Analysis Goals

Core ADL

Quality Extension
Open Questions

Extensible analyses

• How to combine extensions to lower duplicated information?

Analyses

• What does composability for data flows mean?
• What are the limits of the analytical analysis approach?

Modeling

• How to find an appropriate abstraction for runtime configuration?
• Is including runtime configuration via data processors sufficient?
• Can non-experts find the right granularity for the models?
Conclusions

Quality attributes
• Heavily affect business value
• Should be predicted before deployment

Data flows as first class entities
• Allow expressing quality attributes in natural way
• Enable prediction of many qualities

Integration in ADL beneficial
• Reuse of existing models
• Improved comprehensibility

Analyses
• Analytical approach with constraint solving
• Simulative approach
References

- [Ahrendt et al. 2005]

- [Kramer et al. 2014]

- [Reussner et al. 2016]

- [Schmieders et al. 2015]
References

- [Snelting et al. 2014]

- [Strittmatter et al. 2015]

- [Swiderski and Snyder 2004]
### Palladio Modeling Extension

#### Summary

- **Usage profile**
  - Data definitions
  - Binding of data classes to system calls

- **Software components**
  - Data processors
  - New SEFF containing chained data processors

- **Resource environment**
  - Annotations for infrastructure elements

- **Runtime configuration model (new)**
  - Abstractions of runtime configuration
    - Access control
    - Firewall rules