



Fitting Problems

Addressing Architectural Mismatches when Architecting Dependable Systems

Cristina Gacek

Email: cristina.gacek@ncl.ac.uk

27th of November 2008

Overview

- **Motivation**
- **Architectural Mismatches**
- **Architectural Mismatch Avoidance**
- **Architectural Mismatch Removal**
- **Architectural Mismatch Tolerance**
- **Current Challenges**

Software Architecture

*Is the structure or structures of a system, which comprises **software elements**, the externally visible **properties** of those elements, and the **relationships** among them.*

- Architectures are described in terms of:
 - Components
 - Connectors capture the interactions
 - Constraints govern how components and connectors can be composed

Error Confinement

- An architecture is “good” depending on how effectively it can be used to provide means for *error confinement*
 - The structuring of systems should restrict the propagation of errors
 - A good structuring should imply what interactions cannot occur
- An architecture helps
 - Identify components’ behaviour
 - Restrict components’ interactions

Building Complex Systems

- System built from existing components
 - The integrity of the system increasingly depends on the integrity of the components interconnection

- Inevitable that inconsistencies exist between components
 - Analogous to faults in programs
 - Avoided or removed during development time
 - Tolerated during run time

Architectural Mismatches

- Design faults
- Occur when assumptions on the services provided and required do not match
 - Nature of the elements
 - E.g., components being re-entrant or assuming single thread of control
 - E.g., connectors supporting data transfers using a specific transport protocol
 - Global system structure
 - Control flow, data flow, and synchronization issues
 - Organization of the system
 - E.g., layering or distribution

Categories of Architectural Mismatches¹ (I)

□ Data representation

■ Packaging information

- ❖ *Components using different data types for a shared data item.*

■ Semantic information

- ❖ *Components using the same type but with different semantics for a shared data item.*

1. Inspired on DeLine's work [DeLine 1999]

Categories of Architectural Mismatches (II)

□ Data and control transfer

■ Mechanism

■ Directionality

■ Protocol

❖ *One component expects to obtain a shared piece of data by reading a shared variable, whereas another assumes that values for the shared piece of data will be transferred by message passing.*

❖ *One component expects to initiate execution of another component by means of spawning a new thread and continuing execution, whereas the component being initialized assumes that this is happening by means of a call/return.*

Categories of Architectural Mismatches (III)

- State persistence
 - Degree of state retained between interactions
 - ❖ *One component may assume stateful interactions with another, that in turn retains no state information from one interaction to the next.*
- State scope
 - Scope of internal state that other components may affect
 - ❖ *One component may restrict what aspects of its state may be impacted by specific components, that in turn may assume they have full state access.*

Categories of Architectural Mismatches (IV)

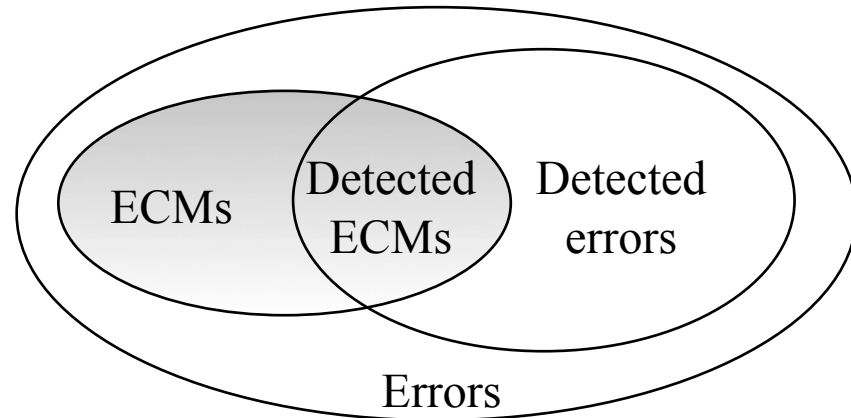
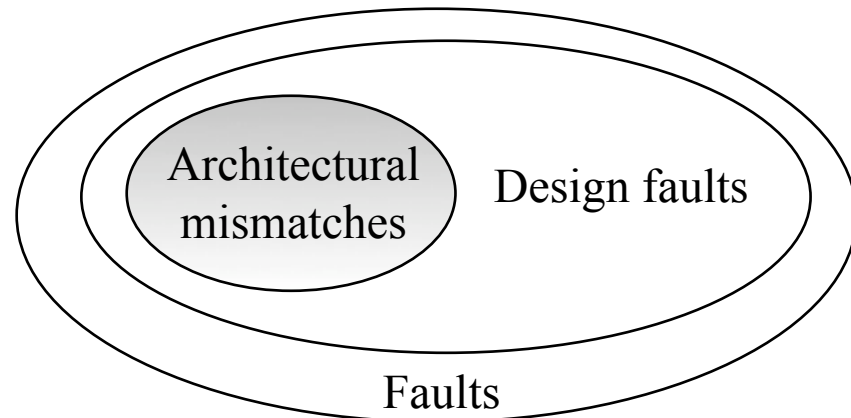
- Connection establishment and release
 - Mechanisms for establishing and releasing connections
 - ❖ *Two components may be connected via one way message passing. The producer assuming that it will be informed when the consumer is no longer there such that the connection can be released, whereas the consumer assumes that the producer uses some other mechanism to poll for listeners and release connections that are no longer needed.*

Categories of Architectural Mismatches (V)

- Dependability meta data
 - Intrinsic dependability related characteristics depicted by various architectural elements. Includes timeliness, failure modes, availability...
 - ❖ *One component (or connector) may assume that another specific component will gracefully handle crash failures, when this is not the case.*

Errors Caused by Mismatches

- Architectural mismatches
 - Occur because of logical inconsistencies among architectural elements
 - Are design faults
- Errors caused by mismatches (ECMs)
 - Latent
 - Detected



Mismatches and Dependability Technologies

- Mismatch avoidance
 - Imposing strict rules on how components should be built and integrated
- Mismatch removal
 - Using static analysis methods and techniques
- Mismatch tolerance
 - Structuring for
 - Error confinement
 - Facilitating ECMs' detection and recovery
 - Hindering the reactivation of mismatches

Overview

- Motivation
- Architectural Mismatches
- *Architectural Mismatch Avoidance*
- **Architectural Mismatch Removal**
- **Architectural Mismatch Tolerance**
- **Current Challenges**

Architectural Mismatch Avoidance

- Imposing strict rules on how components should be built
 - Dictating all relevant characteristics of the individual components and connectors being developed
- Imposing strict rules on how components should be integrated
 - Assumes integrators are aware of incompatibilities
 - Adds code to protect a component or its environment against potential mismatches
 - ❖ *Wrappers*
 - ❖ *Bridges*
 - ❖ *Mediators*

Sample Works on Mismatch Avoidance (I)

- Flexible packaging²
 - Aims at avoiding mismatches introduced by premature packaging decisions
 - Separates functionality from packaging information, deferring decisions on packaging aspects until as late as possible

2. [DeLine 2001]

Sample Works on Mismatch Avoidance (II)

□ Architectural styles

- Aim at providing well formedness rules for systems
- Restrict types of allowed components and connectors, as well as their configurations
- Examples [Shaw and Garlan 1996], [Gacek and Gamble 2008], and many others

Overview

- Motivation
- Architectural Mismatches
- Architectural Mismatch Avoidance
- *Architectural Mismatch Removal*
- **Architectural Mismatch Tolerance**
- **Current Challenges**

Architectural Mismatch Removal

- Using static analysis methods and techniques
 - Detect mismatches during integration of arbitrary components
 - Evaluate options based on component specifications
 - Detect mismatches during architectural modelling and analysis
 - Use components' invariants and services to analyze for architectural conformance

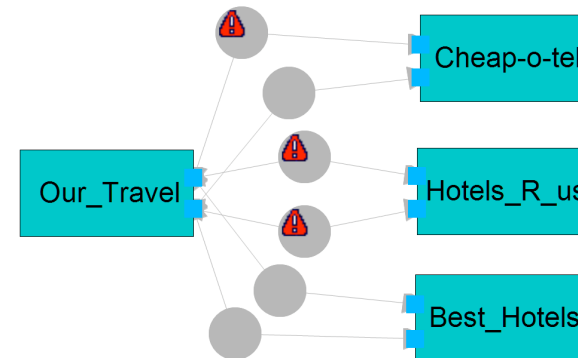
Sample Works on Mismatch Removal (I)

- Architect's Automated Assistant (AAA)³
 - Aims at supporting the exploration of various architectural solutions
 - Presents a differentiation of all architectural features relevant to mismatch detection vs. those that are predefined for specific styles
 - Uses information on components' features, connectors used, and their configuration to highlight potential mismatches
 - Underlying Z model

3. [Gacek 1998]

Sample Works on Mismatch Removal (II)

- Web services in ACME Studio⁴
 - Aims at identifying architectural mismatches in systems built from web services
 - Uses information on components' properties, connectors used, and their configuration to highlight existing mismatches
 - Models in ACME & Armani



4. [Gacek and Gamble 2008]

But...

- It is impossible to localise and correct all architectural mismatches statically
 - Complex applications have complex software architectures, with components interconnected in complex ways
 - Software architects make mistakes
 - Off-the-shelf elements may not expose information on architectural characteristics
 - Software systems may reconfigure dynamically

Overview

- Motivation
- Architectural Mismatches
- Architectural Mismatch Avoidance
- Architectural Mismatch Removal
- *Architectural Mismatch Tolerance*
- **Current Challenges**

Architectural Mismatch Tolerance (I)

- Mismatch tolerance
 - ECMs processing
 - Detection, diagnosis, and recovery
 - Mismatches treatment
 - Diagnosis, isolation, and reconfiguration
- Systems should be structured for
 - Error confinement
 - Facilitating ECMs' detection and recovery
 - Hindering the reactivation of mismatches

Architectural Mismatch Tolerance (II)

- Two abstraction levels to consider
 - **Architectural level** where the mismatches are introduced
 - **Execution level** where the ECMs are detected and recovered from
- Additional information is needed
 - To associate an error with a mismatch
 - Distinguish ECMs from other system errors
 - To diagnose a mismatch
 - Determine its nature and location

Architectural Mismatch Tolerance (III)

- Why specify mechanisms at the architectural level?
 - Nature and context of the mismatch is not lost
 - Different kinds of mismatches require different detection mechanisms and fixes
 - There are no general runtime mechanism to deal with a wide range architectural mismatches
- Mismatch activation depends on system state and the presence of inconsistent assumptions

ECM Processing

- Detection
 - Identifies erroneous state and determines whether it is an ECM
 - Requires additional run time information
- Diagnosis
 - Assesses system damages caused by detected ECM
- Recovery
 - Brings the system to an ECM-free state
 - Depends on ECM's characteristics

Mismatch Treatment (I)

□ Diagnosis

- Determine cause of ECM (nature & location)
- Identify architectural elements that failed & how they failed
- Information needed
 - Erroneous state
 - Overall system state
 - Architectural elements, their configurations, and properties

Mismatch Treatment (II)

□ Repair

- Dynamic reconfiguration
- May require redundant architectural elements with known diverse architectural features
- Remove, add, and/or replace architectural element(s)

Sample Works on Mismatch Tolerance (I)

- Architectural Mismatch Tolerance⁵
 - Aims at motivating the need for architectural mismatch tolerance
 - Uses differentiations between style-specific and application-specific properties of architectures and their elements
 - Suggests different mechanisms for ECM processing and mismatch treatment using style and application specific differentiations

5. [de Lemos, Gacek, and Romanovsky 2003]

Sample Works on Mismatch Tolerance (III)

- Architectural Monitoring⁷
 - Aims at enabling run time monitoring of events or states that are architecturally relevant
 - Provide a mapping solution integrating primitive events and architectural events

7. [Dias and Richardson 2003 and 2005]

Sample Works on Mismatch Tolerance (IV)

- Adapting Service Requests⁸
 - Aims at facilitating the invocation of services (SOA) with mismatched operation names, data representation, or message exchange pattern
 - Uses previously defined mappings between expected and provided interfaces to mediate communications at run time

8. [Cavallaro and Di Nitto 2008]

Overview

- Motivation
- Architectural Mismatches
- Architectural Mismatch Avoidance
- Architectural Mismatch Removal
- Architectural Mismatch Tolerance
- *Current Challenges*

Current Challenges (I)

□ Architectural Mismatches Avoidance

- Not all components expose their characteristics relevant for mismatch detection
- Impossible to guarantee that all rules are being followed

□ Architectural Mismatches Removal

- Not all components expose their characteristics relevant for mismatch detection
- Incompatibilities may be impossible to identify

Current Challenges (II)

- Architectural Mismatches Tolerance
 - How to efficiently monitor a system at run time for mismatch tolerance
 - How to map between architectural models and execution level information
 - What are appropriate notations to represent the various pieces of information
 - How to associate an error with a mismatch
 - How to address scalability with respect to ECM processing and mismatch treatment

Current Challenges (III)

- Architectural Mismatches Tolerance (cont'd)
 - How to perform dynamic reconfiguration⁹
 - Expressing and selecting policies
 - Are actions taken from a centralized or a decentralized point of view
 - Effective provision of degraded services until reconfiguration is accomplished
 - Avoidance of “thrashing”
 - Timeliness

9. Similar to concerns with self-adaptive systems [B.H.C. Cheng et al. 2008]

References (I)

--, Architecting Dependable Systems, <http://www.cs.kent.ac.uk/people/staff/rdl/ADSFuture/>

R. DeLine, “A catalog of techniques for resolving packaging mismatch,” *Proc. Symposium on Software Reusability*, pp 44-53, 1999

R. de Lemos, C. Gacek, and A. Romanovsky, “Architectural Mismatch Tolerance,” *Lecture Notes in Computer Science*, vol. 2677, pp 175-196, Springer, 2003

C. Gacek and R. de Lemos. “Architectural Description of Dependable Software Systems,” *Structure for Dependability: Computer-Based Systems from an Interdisciplinary Perspective*, D. Besnard, C. Gacek, C. Jones (Eds.), pp 127-142, Springer-Verlag, 2006

R. DeLine, “Avoiding Packaging Mismatch with Flexible Packaging,” *IEEE Transactions on Software Engineering*, vol. 27, no. 2, pp 124-143, 2001

M. Shaw and D. Garlan, *Software Architecture: Perspectives on an Emerging Discipline*, Prentice-Hall, 1996

C. Gacek, *Detecting Architectural Mismatches during Systems Composition*, PhD Dissertation, Centre for Software Engineering, University of Southern California, USA, 1998

References (II)

- C. Gacek and C. Gamble, “Mismatch Avoidance in Web Services Software Architectures,” *Journal of Universal Computer Science*, Special Issue on Software Components, Architectures and Reuse, vol. 14, no. 8, pp. 1285-1313, 2008
- R.de Lemos, P. A. de C. Guerra, and C. Rubira, “A Fault-Tolerant Architectural Approach for Dependable Systems,” *IEEE Software*, vol. 23, no. 2, pp. 80-87, 2006
- M. S. Dias and D. J. Richardson, “The Role of Event Description in Architecting Dependable Systems,” *Architecting Dependable Systems*, LNCS 2677, pp. 150-174, 2003
- M. S. Dias and D. J. Richardson, “Adaptable Analysis of Dependable System Architectures through Monitoring,” *Architecting Dependable Systems III*, LNCS 3549, pp. 122-147, 2005.
- L. Cavallaro and E. Di Nitto, “An approach to adapt service requests to actual service interfaces,” *Proc. International Workshop on Software Engineering for Adaptive and Self-Managing Systems*, pp. 129-136, 2008
- B.H.C. Cheng et al., “Software Engineering for Self-Adaptive Systems: A Research Road Map (Draft Version),” 2008