USC

USC Viterbi
School of Engineering

Architectural Conformance in Message-Based Systems

Daniel Popescu

Nenad Medvidovic

Motivation

- Dependability properties analyzed at architectural level

- Mismatches between architectural and implementation abstractions
    - Configuration of components and connectors vs. objects and packages

- Ad-hoc implementation causes architectural drift

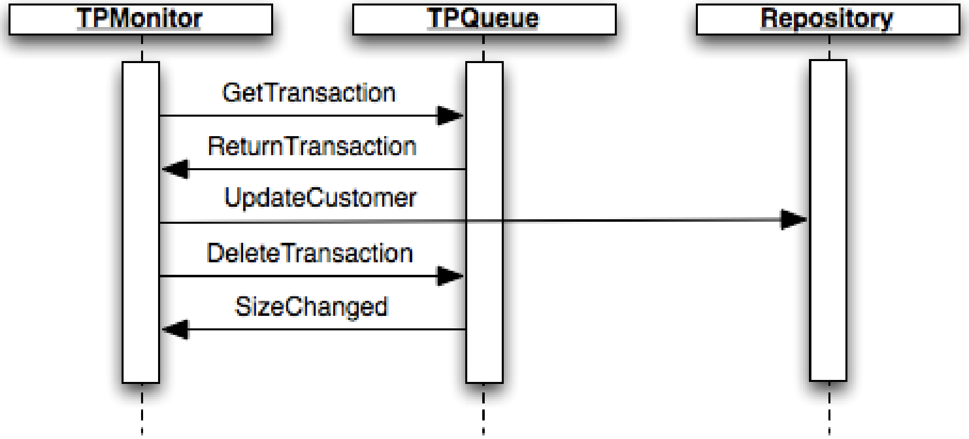
    → Analysis based on prescriptive architectural models cannot be assured

- Techniques ensuring static prescriptive architectures with static implementation match
    - E.g., Reflexion models or architectural implementation frameworks

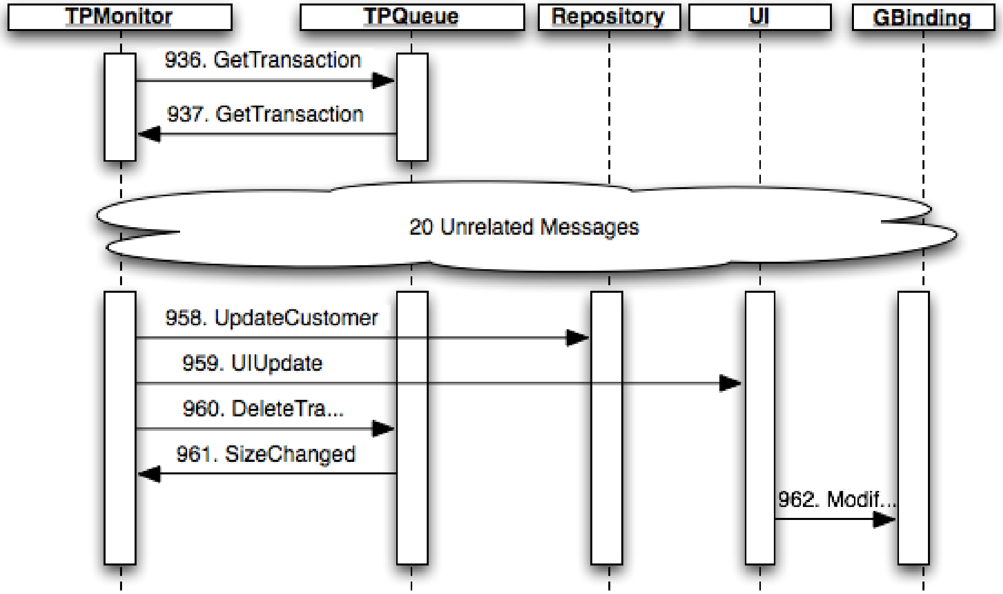
- Behavioral conformance required for assurance of dependability properties
    - How can we assess whether *sequences of event*s exchanged among implemented *concurrent* components comply to *prescribed sequences of events*?

Checking Behavioral Conformance

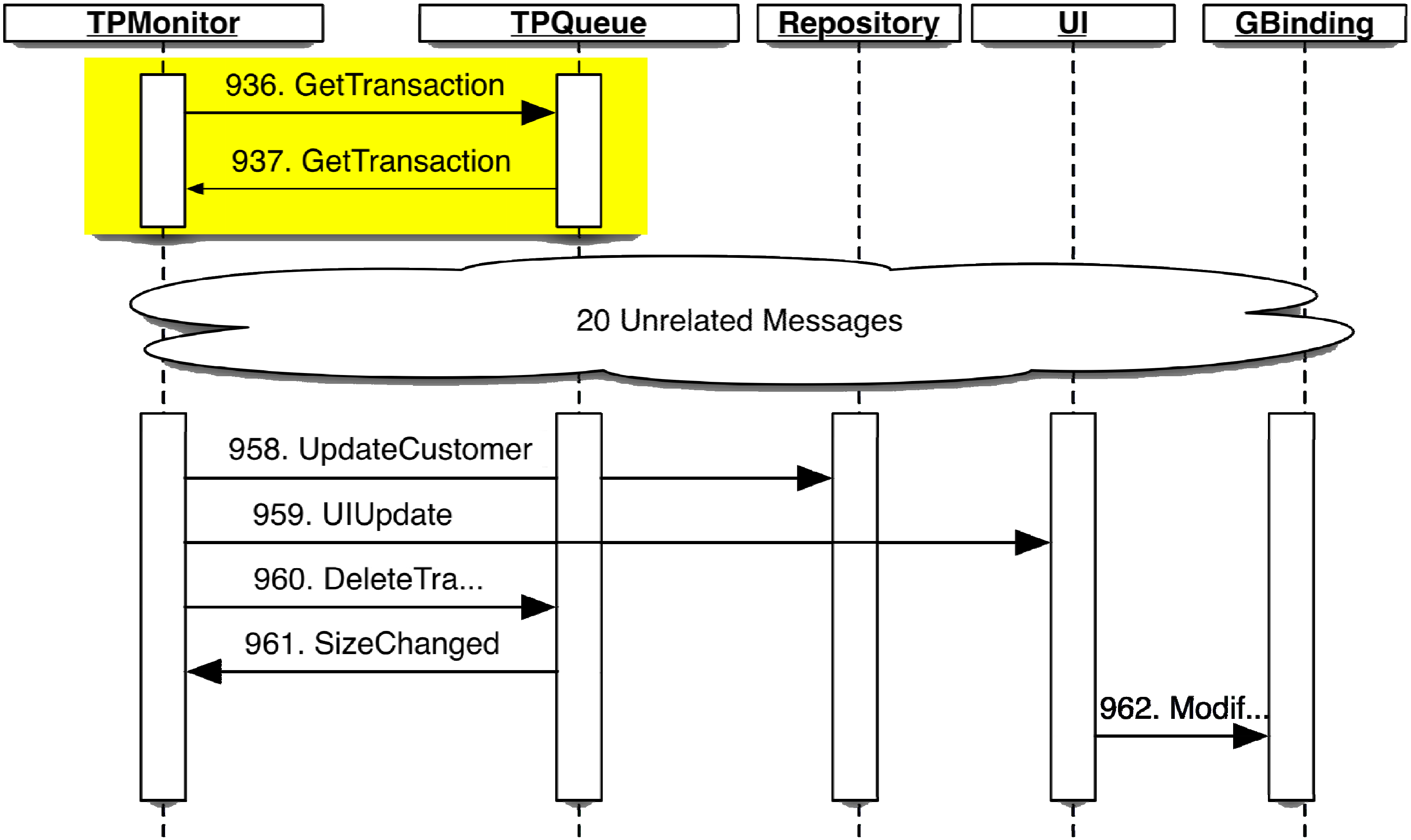
- Prescriptive Sequence of Events



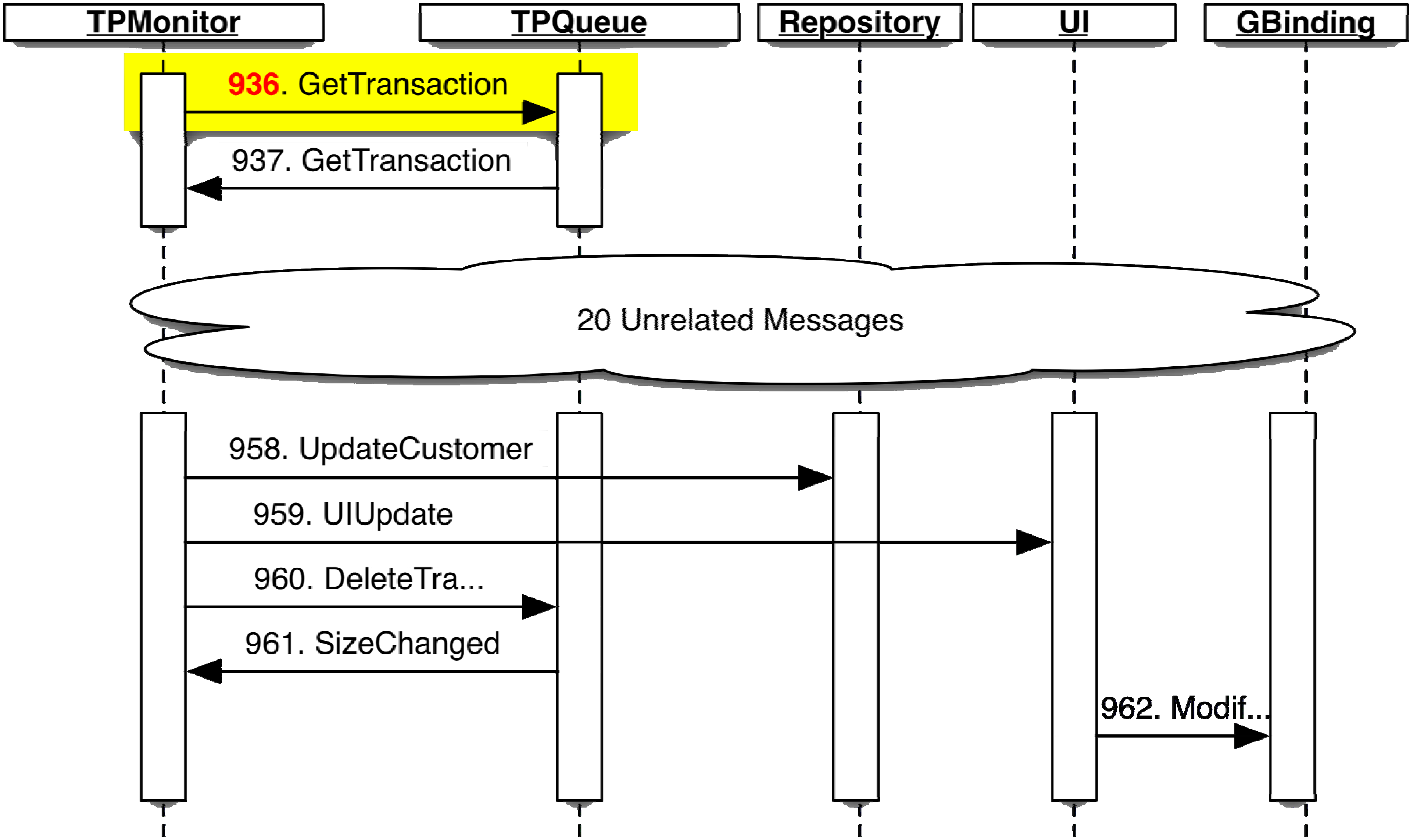
- Recorded Message Trace



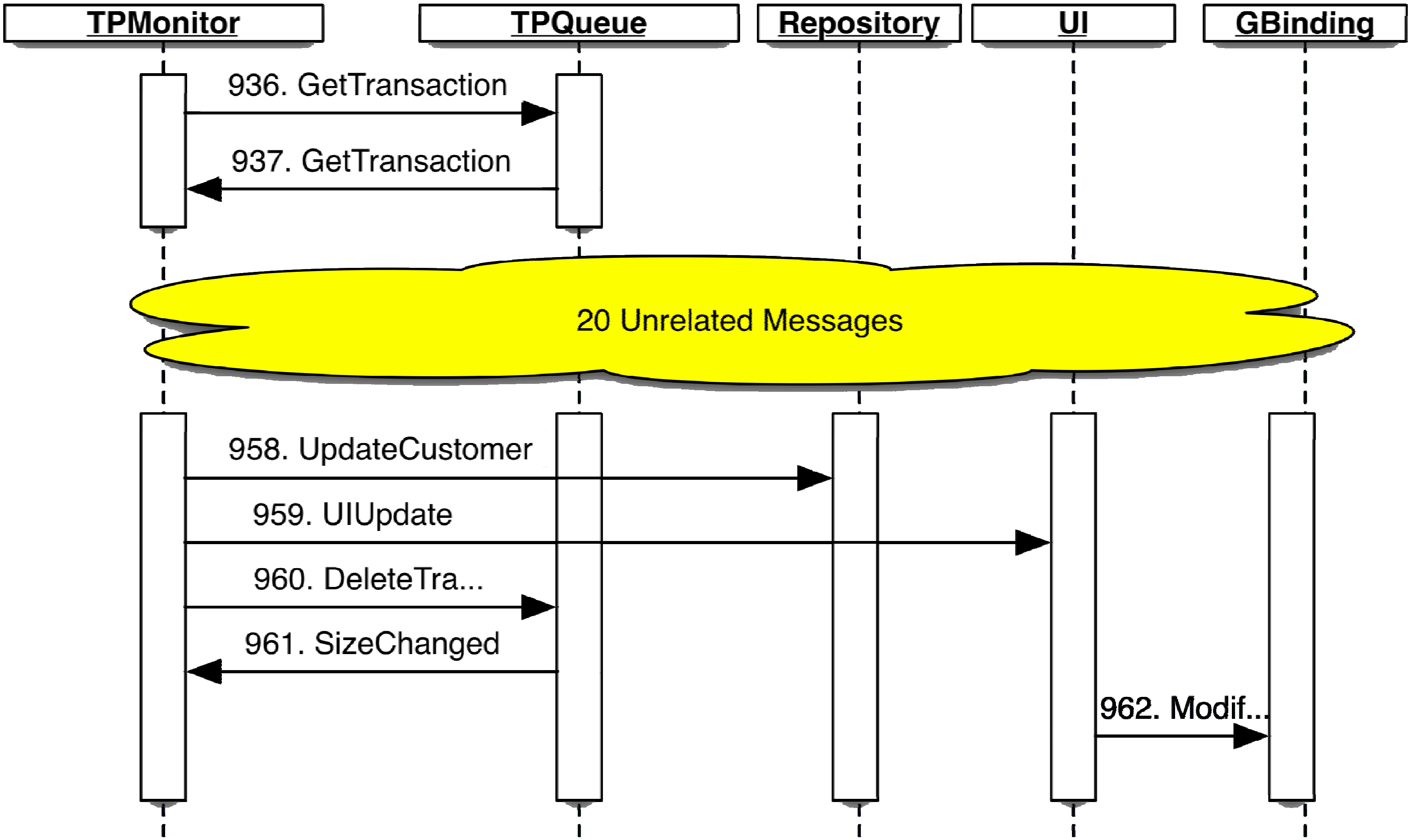
Renamed Events



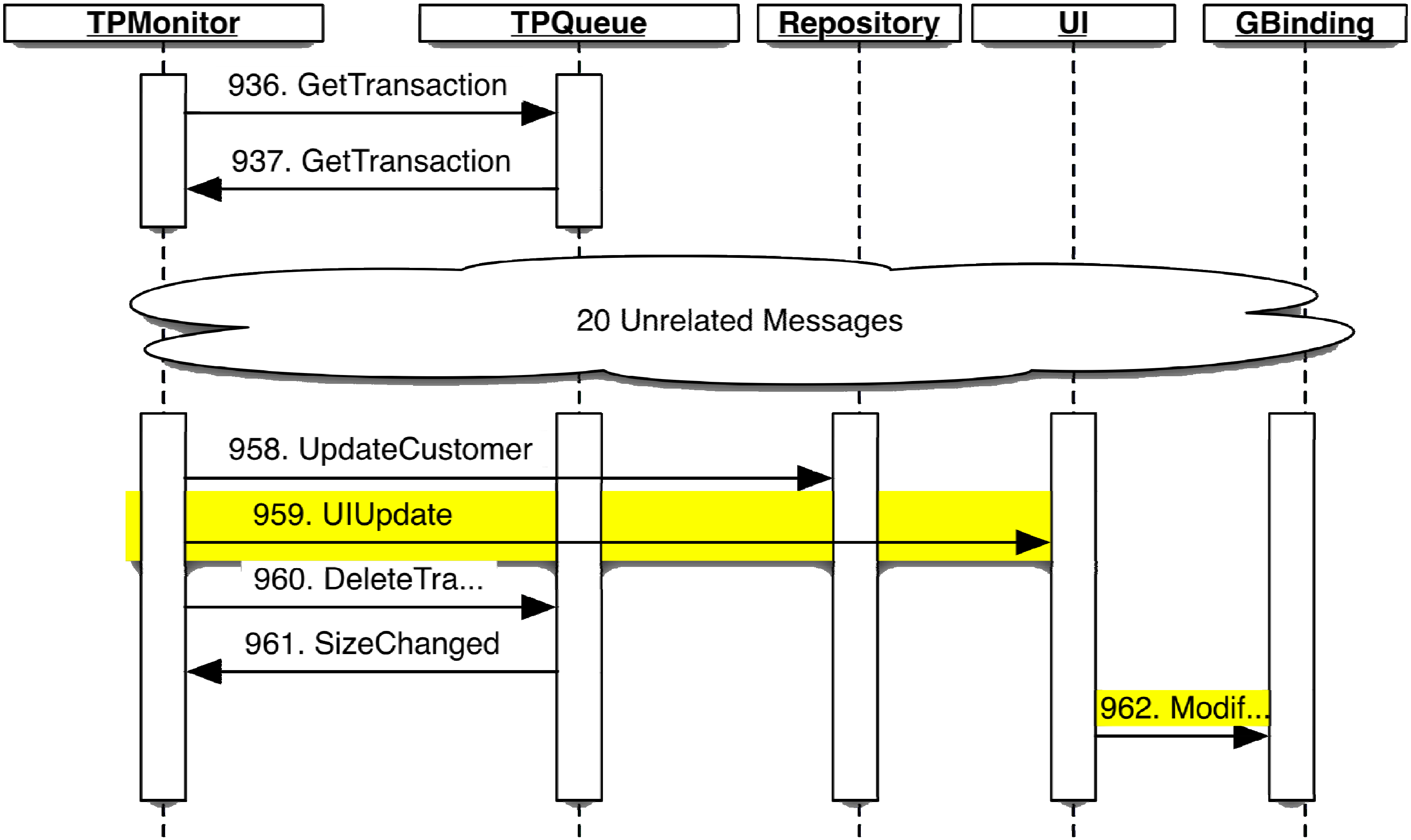
Position in the Trace



Concurrent Communication



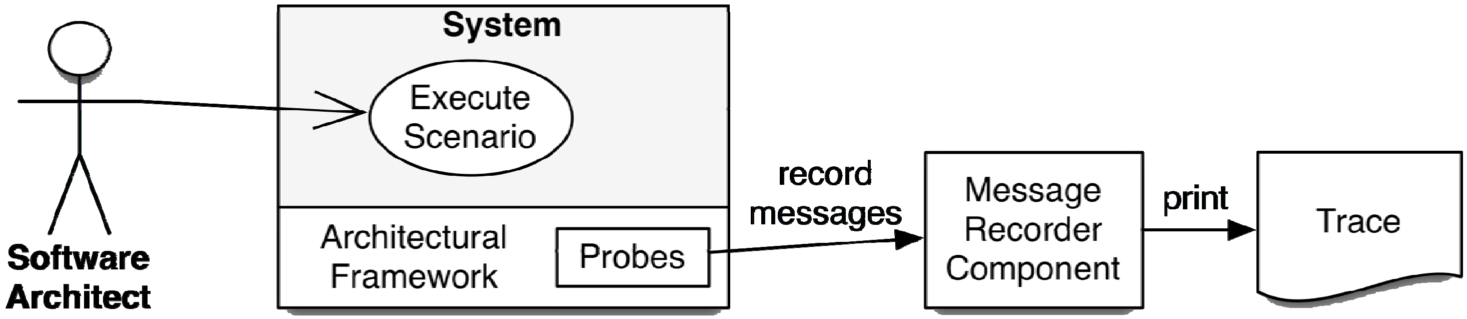
Related Non-Prescribed Messages



Implementation Platform

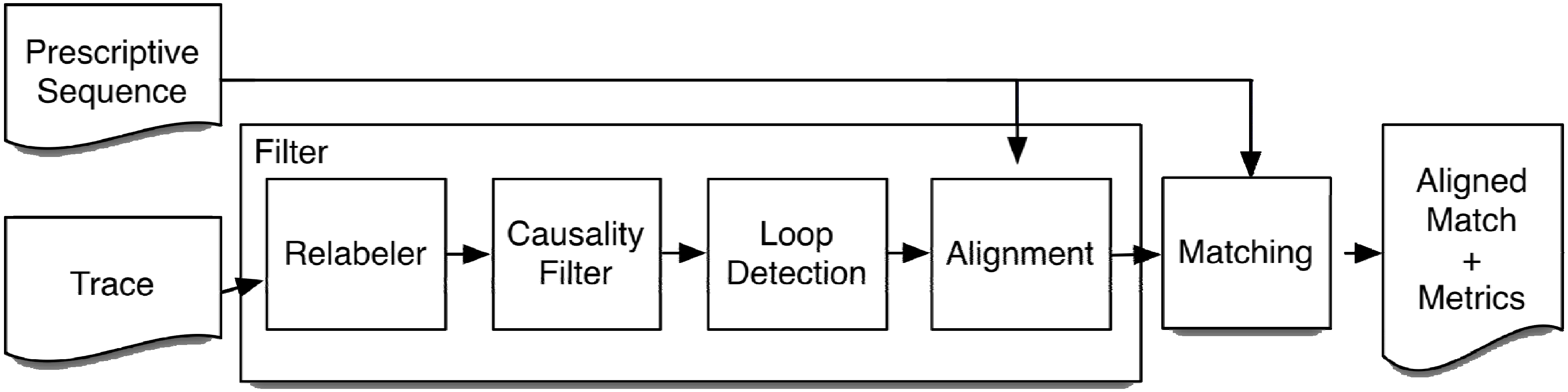
- Studied systems implemented using architectural event-based implementation frameworks (Prism-MW and c2.fw)
  - Support architectural abstractions (components, connectors, configurations, ports, ...)
  - Support concurrent architectural components
- Architectural communication helps reduce the trace size explosion problem

Trace Extraction

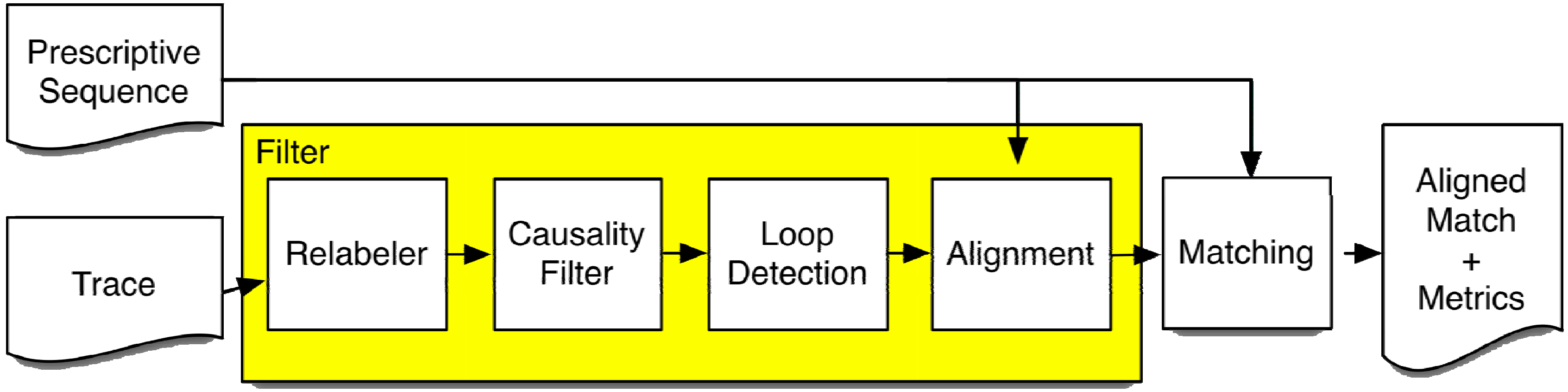


- Software architect executes scenario
- Software probes
  - At the communication ports of each component
  - Extract architectural communication events
  - Extract event causalities via heuristic
    - E.g., event A causes a component to emit event B and event C
- Message Recorder Component records trace

Conformance Checking Approach Overview

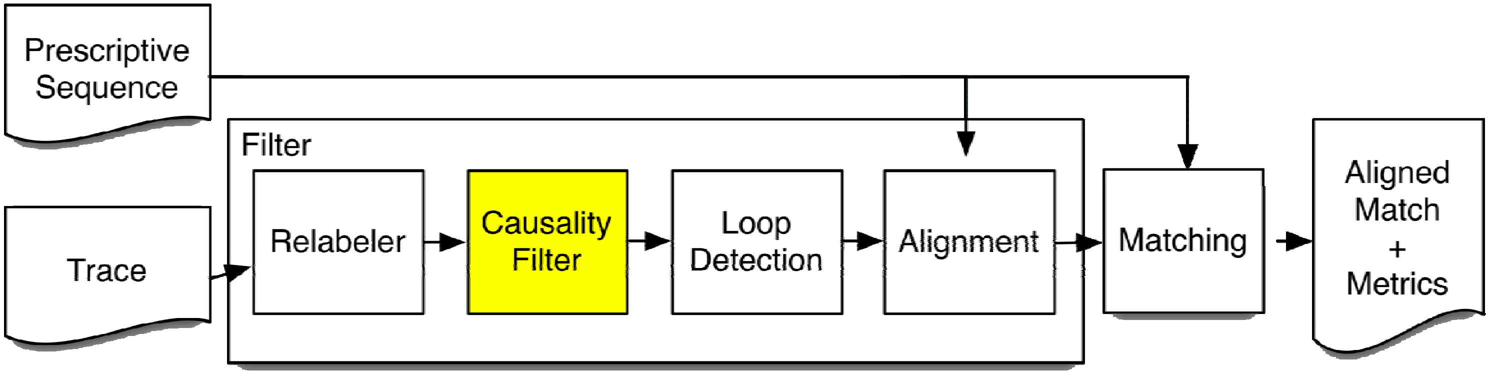


Filtering



- Reduce trace size
  - Traces usually substantially larger than prescriptive sequences
    → E.g., stock ticker scenario caused over 1000 events
  - Try to minimize information loss

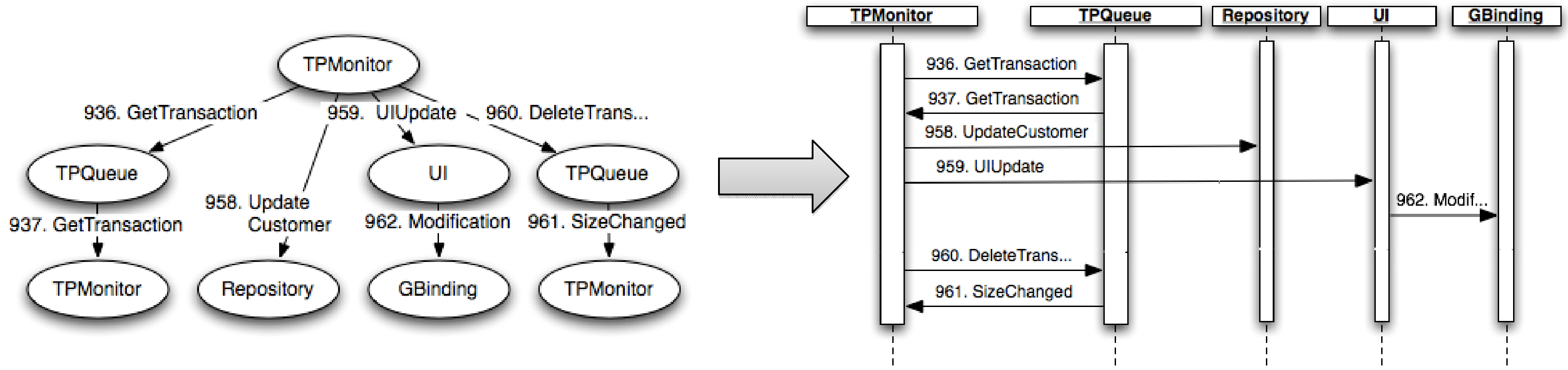
Causality Filtering



Causality Filtering

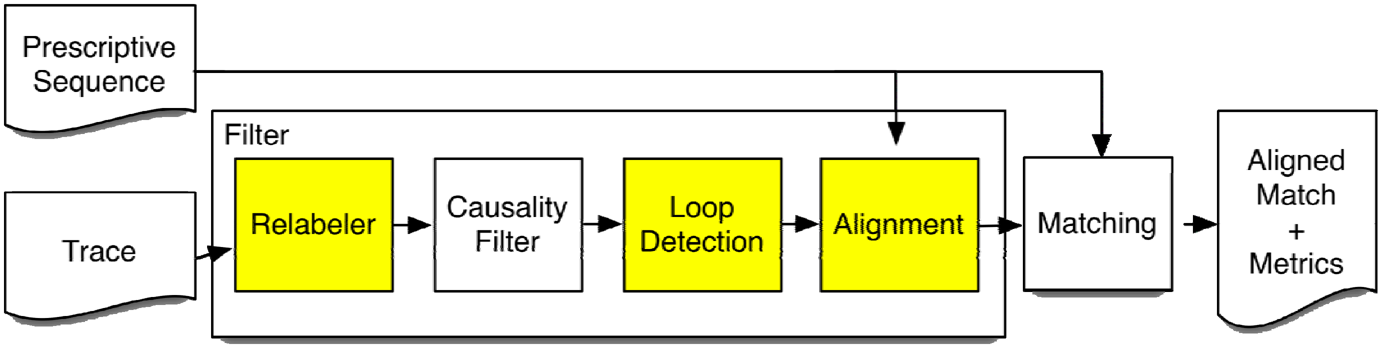
Causality Tree

Split and Reordered Trace

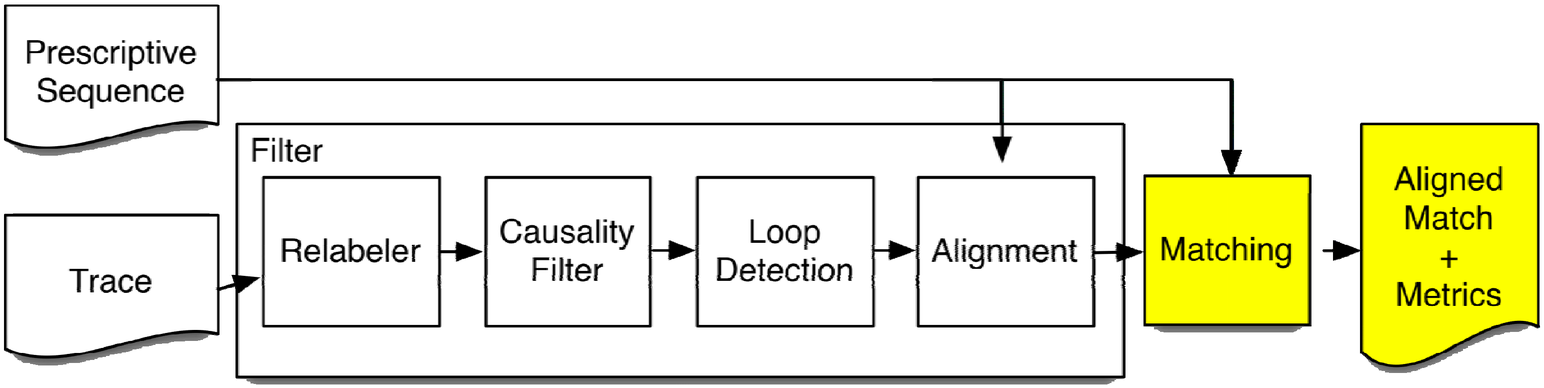


- Concurrency → trace containing intertwined sequences
- Causality filtering helps untangle intertwined sequences
- Causality filtering
  - Uses the heuristically extracted causality relationships
  - Identifies causally connected sequences
  - Removes events of other concurrent use cases
  - Optimizes order of event sequences

Trace Filtering



- Relabeling
  - E.g., changing names of token event instances to generic “token” event
- Loop Detection
  - Prescriptive sequence do not contain loops
- Excerpt Detection and alignment
  - Trigger messages help identify relevant trace excerpt


Prescriptive Sequence
Trace
Filter
Relabeler
Causality Filter
Loop Detection
Alignment
Matching
Aligned Match + Metrics

Sequence Matching

- After filtering: noise messages and errors may still exist in the trace

- Implementation-level decisions can affect the trace

  → Exact string matching would almost always cause a failed matching

  → We use approximate pattern matching algorithm based on Levenshtein distance

- Final output
  - Levenshtein distance
  - Prescriptive-to-length ratio
  - Prescriptive sequence and the trace aligned to each other

Conclusion and Future Work

- Contributions
  - Error-tolerant conformance technique for architectural behavioral descriptions
  - Reduction of trace size explosion problem
    - Focus on the architectural communication
  - Causality Filtering
- Future Work
  - More thorough experiments
  - Expansion to more complex prescriptive event sequence modeling constructs
  - Different implementation technologies and frameworks
    - Interaction protocols
    - Synchronous implementation frameworks