An automated interference identification approach

Towards INFoRMED - INterFerence Removal MEthoD

Christoph Dropmann
Christoph.dropmann@iese.fraunhofer.de
We define an interference as a cascading failure via a shared resource.

Typical example: Memory corruption, CPU monopolization.

Threat to an integrated system (see AUTOSAR or IMA).

Can lead to undesired criticality inheritance.
Motivation

- Current situation
  - Movement towards mixed critical systems and more complex platforms
  - Methods and techniques deal with interferences caused by:
    - Dependencies via Scheduling & Memory
    - Physical resource connections (power and temperature)

- Problem
  - Complex computation platforms contain heterogeneous services
  - Little guidance for interferences caused by
    - Logical dependencies within a service?

- Our Solution Idea
  - Provide an automated approach for interference analysis of services
  - Increases confidence in the completeness of the analysis and
  - reduces the impact of human skill and judgment on the analysis quality
Solution Overview - INFoRMED

Service

 legacy

 << automated >>
 model import

 << systematic >>
 manual modeling

 new

 << systematic >>
 model enrichment

 << automated >>
 interference analysis

 protected?

 << semi-automated >>
 protection assignment

 Protection Mechanisms : Protection Model

 Modelling Language

 Analysis Method

 Segregation Method based on Protection Templates

 Interference Aspects

 Segregated Service

 << automated >>
 protection case generation

 yes

 no
Solution Overview - INFoRMED

Modeling Language
Analysis Method
Interference Aspects

Segregation Method based on Protection Templates

Protection Mechanisms: Protection Model

Service

Segregated Service

<< automated >>
protection case generation

<< semi-automated >>
protection assignment

<< systematic >>
model enrichment

<< automated >>
interference analysis

<< systematic >>
model import

<< systematic >>
manual modeling

<< automated >>
model import

protected?
no
yes
Solution – Service Interference Identification (1)

- Interference scenarios are analyzed for a shared service (e.g. a platform service for redundant execution or a memory management service, ...)

- The interference identification bases on:
  - An interference specification template that allows a structured description
  - The interference aspects that defines different, independent classes of interference
  - The modeling language that allows an automated analysis
  - The analysis algorithm to extract the interference scenarios

![Diagram of shared service modeling with interference scenario and template analysis]
Solution – Service Interference Identification (2)

Interference Aspects

- Spatial
  - Service Object Corruption
  - Service Object Over-Allocation

- Temporal
  - Service Overutilization
  - Concurrent Service Access Delay

- Behavioral
  - Service Misuse
  - Service Configuration Corruption
Interference Aspects - Spatial

- **Sharing Preconditions / Intention**
  - The service or parts of it are not supposed to be shared
  - I.e.: Access to the service, or service part is limited to one, or a set of service users

- A **Service Object Corruption** occurs if
  - “… an unauthorized service user accesses or manipulates a service or service object”
  - Example: Corruption of a data block of a memory management service …

- An **Service Object Over-Allocation** occurs if
  - “… a service user allocates more service objects than specified”
  - Requires that service objects can be allocated during run-time
  - Example: Over-Allocation of memory (e.g. via malloc), queue entries, …
Interference Aspects - Temporal

- **Sharing Preconditions / Intention**
  - The service is shared over time
  - I.e.: At any point in time, only one user can access the service

- **A Concurrent Service Access Delay** occurs if
  - "... a service user access a synchronization mechanisms of a service more often than expected"
  - Requires that the service can be accessed from applications in parallel. As a result the execution time of the waiting/effectected service user increases
  - Examples: Using spinlocks in a service implementation, ...

- **A Service Overutilization** occurs if
  - "... a service user’s maximum contention delay is exceeded”
  - Requires that service accesses are arbitrated dynamically. As a result response time of affected service users increases
  - Examples: Overutilization of a service’s job queue
Interference Aspects - Behavioral

- Sharing Preconditions
  - The service, or parts of it are shared
  - I.e.: Different users are allowed to access the service

- A Service Misuse occurs if
  - “… a service user erroneously changes implicit the functional behavior of a shared service”
  - Requires that a service is capable of performing actions that have a service-wide or even system-wide effect
  - Example: an application termination leads to an undefined service state

- A Service Configuration Corruption occurs if
  - “… a service user erroneously changes the configuration of a shared service”
  - Requires that a service has configuration parameters changeable at runtime
  - Example: Reconfiguration of a device driver service
Solution Overview - INFoRMED

Service

 legacy

 << automated >>
 model import

 new

 << systematic >>
 manual modeling

 << systematic >>
 model enrichment

 << automated >>
 interference analysis

 << semi-automated >>
 protection assignment

 Segregated Service

 << automated >>
 protection case generation

 yes

 no

 protected?

 Protection Mechanisms: Protection Model

 Modelling Language

 Analysis Method

 Segregation Method based on Protection Templates

 Interference Aspects
Solution– Modelling Language Basic Idea (1)

UML class model of a service:

Service, e.g. RingBuffer

Attribute e.g. Data

Operation e.g. peek(RingBufId, *Data)

OutputParameter

InputParameter

Enrichment:
- Relations
- Meta scenarios

1. uses failure propagation failure

application 1 (causative element)

application N (affected element)
Solution – Modelling Language Basic Idea (2)

class Peek
  «out» dataPointer
  «in» dataSize
  «in» RingbufferId

«Operation» peek
  «out» dataPointer
  «in» dataSize

«Composite Attribute» Ringbuffer
  «Attribute» Data
  «Attribute» Count

«Modification» CopyData
  «use» «provides data»
  «use» «selects»
  «use» «modifies»
  «use» «selects»

Interference Channels
Summary, Limitations and Future Work

- **Summary**
  - We present an automated service interference analysis approach
  - Proposed benefits are efficient service segregation and reduction of human skill- and judgment

- **Limitations**
  - Functional dependencies between service users are currently not considered (synchronization issues)
  - Correctness of a service model depends on human skills

- **Future Work**
  - Completion of the modelling language
  - evaluate the analysis with different services
  - Integration of protection strategies