Adding Safeness to Dynamic Adaptation Techniques

Betty H.C. Cheng
Software Engineering and Network Systems Laboratory
Michigan State University
http://www.cse.msu.edu/SENS

Authors: J. Zhang, Z. Yang, B. H.C. Cheng, and P. K. McKinley

ACKNOWLEDGEMENTS: This work has been supported in part by grants: NSF EIA-0000433, CDA-9700732, CCR-9901017, EIA-0130724, ITR-0313142, Department of the Navy, and Office of Naval Research under Grant No. N00014-01-1-0744.
**RAPIDware Project**

- Ongoing project in SENS Laboratory
- Funded by U.S Office of Naval Research
  - Critical Infrastructure Protection /Adaptable SW Program
- **Goal:** Software (middleware) that can protect itself from:
  - Hardware and software component failures
  - Changing environmental conditions
  - Changing requirements (e.g. security policies)
  - Malicious entities
- **Applications:**
  - Dynamic power management
  - Dynamic error correction for data transmission/receipt
  - Dynamically changing security algorithms and policies
  - Dynamic introduction of fault-tolerant capabilities
Outline

• Dynamic adaptation
• Safe Adaptation
• Example Application
Dynamic Adaptation

• At run time, adapt software in response to changes in:
  – environment, requirements, etc.

• Significant work in:
  – Adaptation mechanisms
  – Programming language extensions
  – Architectural description languages

• Correctness/Accurayse Issues:
  – Adapted system provides correct functionality
  – Safeness: During adaptation process, no unexpected or undesirable results
Key Concepts

Assumptions:
– A distributed system is modeled as a set of communicating components running on one or more processes.
– Adaptive actions: insert, remove, or replace SW elements

• **Atomic communication:**
  – An interaction, either within a component or between components, that cannot be interrupted.
  – Otherwise, it would potentially yield erroneous or unexpected results.

• **Dependency invariants:**
  – relationships among the components that should be held true throughout the program’s execution.

Safe adaptation process:
• Does not interrupt atomic communications.
• Does not violate dependency invariants.
Features

• Use dependency analysis to determine safe states for a given adaptive action

• Centralized management of adaptations,
  – Enable optimizations of adaptive actions

• Roll back mechanism when encounter failures during adaptation process
Safe Adaptation Process

1. Construct minimum adaptation path (given a source and a target configurations).
   (1) **Construct safe configuration set.**
   (2) **Construct safe adaptation graph:**
       vertices are safe configurations and arcs are adaptive actions.
   (3) **Assign a cost value to each arc.**
       (e.g. packet delay caused by the action)
   (1) **Search for minimum safe adaptation path (MAP):**
       path with minimum cost from the source to the target.

2. Manage adaptation process.
   • Components are reset to safe states before adaptation.
   • Blocking is introduced only when it is necessary to ensure safeness.
   • Adaptation process can roll back if encounter failure during process.
Safe Adaptation Process

Adaptation request

Adaptation manager

Construct safe adaptation graph

Construct MAP

Coordinate

Adaptation agent

………

Adaptation agent

 Coordinate

………

Coordinate
Video Streaming Case Study

- MetaSocket [Sadjadi et al]:
  - chain of data stream filters and a Java socket
  - Alter behavior through filter insertion, removal, and replacement.

- Video streaming example
  - Video server: Sends data streams through a MetaSocket
    - A web camera captures video.
    - Video stream is sent to clients through a multicasting MetaSocket.
  - Video clients: Receive data streams through MetaSockets
    - A handheld computer.
    - A laptop computer.
  - Server and clients are connected with wireless networks
Filters available in the MetaSockets

**Server**
- E1: DES 64bit Encoder
- E2: DES 128bit Encoder

**Laptop Client**
- D4: DES 64bit Decoder
- D5: DES 128bit Decoder

**Hand-held Client**
- D1: DES 64bit Decoder
- D2: DES 64/128bit Decoder
- D3: DES 128bit Decoder
Video Streaming Case Study

• **Safe conditions:**
  • *Safe states:* System states in which, adaptive actions do not interrupt atomic communications.
    • Encoder: Not in the middle of encoding a packet.
    • Decoders: No in-flight packet for the decoders to be removed.

• **Dependency invariants:**
  • Collaboration constraints: Each encoder requires the corresponding decoder.
  • Resource constraints: The hand-held device does not support two decoders simultaneously in the device.
  • Security constraints: All packets should be encoded with either 64-bit or 128-bit encoder.
Video Streaming Case Study

• Adaptation goal:
  Reconfigure system
  – From: DES 64-bit encoder/decoders
  – To: DES 128-bit encoder/decoders
  in order to "harden" security at run time
Unsafe Adaptation Scenarios

• **Interruption of atomic communication:**
  – Replace the encoder while it is encoding a packet.
    • Effect: inconsistent results
  – Replace encoder and decoders simultaneously:
    • Effect: In-flight packets will not be decoded.

• **Violation of dependency invariants:**
  – First remove 64-bit DES encoder/decoders then insert 128-bit DES encoder/decoders:
    • Effect: Violates security constraints.
  – First insert 128-bit DES encoder, then insert 128-bit DES decoder:
    • Effect: Violates collaboration constraints.
Video Streaming Case Study

- Use 7-bit vector to represent configuration: (D5,D4,D3,D2,D1,E2,E1)
- Vertices are safe configurations:
  - Source: (0100101)
  - Target: (1010010)
- Arcs are adaptive actions:
  - “+”: insertion
  - “-”: removal
  - “->“: replacement
  - Numbers indicate costs
- MAP: red path identified by safe adaptation process
- Adaptive actions are performed in safe states of system.
Conclusions

- **Safeness**
  - Adaptation process is safe with respect to:
    - not violating dependency invariants and
    - not interrupting atomic communications.
- **Allows for choice and optimization among multiple safe adaptation paths**
- **Supports roll-back mechanism in case of failure during adaptation process**
- **Future work:**
  - Investigating approximation algorithms for MAP
  - Cost measures for adaptive actions
Questions/Discussion

- Acknowledgements:
  - Sandeep Kulkarni, Karun Biyani
  - Other SENS faculty and students

- Supporting grants:
  - NSF: EIA-0000433, CDA-9700732, CCR-9901017, EIA-0130724, ITR-0313142
  - ONR: research grant #: N00014-01-1-0744
Related Work

• Kramer and Magee: Conic and Darwin [1,2]
  – Use *architectural description language* to model the system connection.
  – Separate communication from computation.
  – Dynamically connect or disconnect components.
  – Use *LTSA* to check adaptation models created with *FSP*.

• Appavoo, and colleagues: Hot Swapping [3]
  – *Quiescent* states are the states when it is safe to perform hot-swapping.
  – Use *generation counts* to determine quiescent states.
  – Component state transfer protocols are selected by *transfer negotiation protocol*.
Related Work

• **Schlichting et al: Cactus [4]**
  – *Composite components* are composed of multiple *micro-components*.
  – The composite component can be reconfigured by altering its component micro-components.
  – It uses fuzzy logic to deal with change coordination.
  – It uses *graceful adaptation* process to perform adaptive actions.

• **Taylor, Medvidovic and et al: Chiron-2 and ArchStudio [5]**
  – C2 is layered ADL.
  – Substrate independent and implicit invocation facilitates dynamic insertion, removal, and replacement of components.
  – Systems can be reconfigured in three ways:
    • *Argo*: manipulates the model graphically.
    • *ArchShell*: Use command line to manipulate the system configuration
    • *Extension Wizard*: execute modification script on the end-user's system.
Related Work

• Kulkarni et al [6]
  – safely composing distributed fault-tolerance components at run time.
  – use a spanning tree to pass adaptation messages.
  – uses a reset mechanism to block computations during the recomposition process.
References


References


Motivation

- **Dynamic adaptation is the trend:** software systems must adapt their behavior to changing conditions.

- **Examples warranting dynamic adaptations:**
  - Dynamic introductions of new strategies.
  - Quick responses to security threats.
  - Switching to certain execution mode to save battery life.
  - Insertions of encryption layers to network protocol stack.

- **Dynamic adaptation is prone to errors.**
  - Formalism: Unless adaptive software mechanisms are grounded in formalisms, the resulting systems will be prone to errant behavior.
  - Safe dynamic adaptation separates the safeness issue from the adaptation mechanism, and thus provides the basis for formal reasoning about the adaptation behavior.