Evaluation of software aging in component-based Web Applications subject to soft errors over time

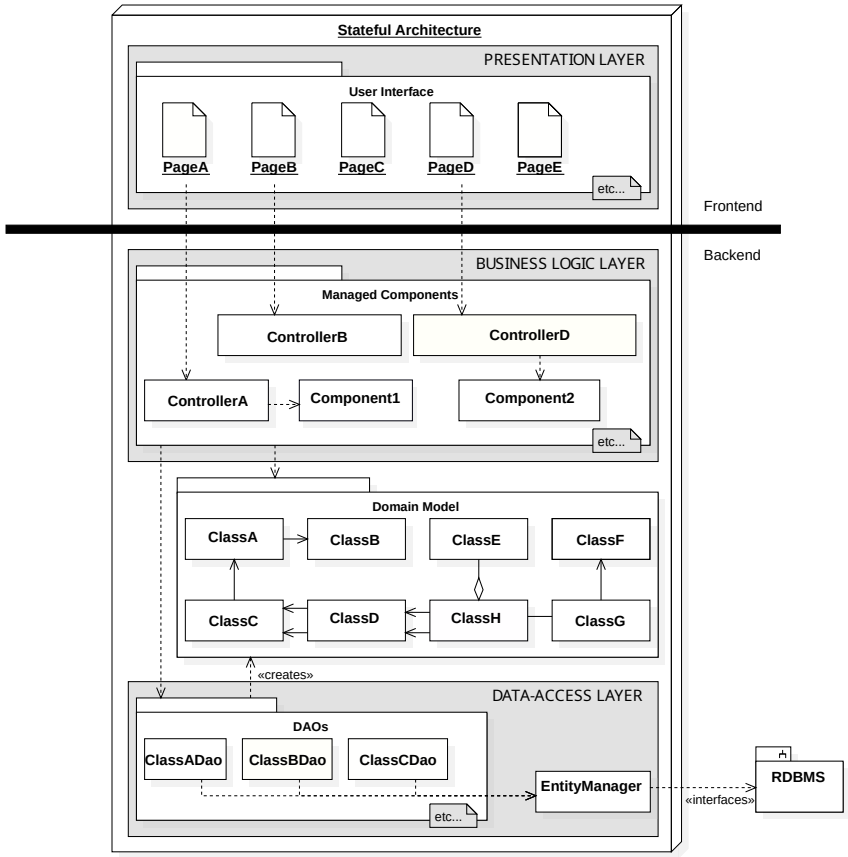
Jacopo Parri, Samuele Sampietro, Leonardo Scommegna, Enrico Vicario
Department of Information Engineering, University of Florence

WoSAR'21 - October 2021

- this is about:
  - How Web Applications maintain in-memory elements causing ARBs
  - How lifecycle management can act as sw micro-rejuvenation
  - How policies of lifecycle design can affect the overall robustness to sw aging
  - Experimental Environment simulating errors activated over time
  - Complex tools developed to observe the phenomenon and open for extensions

Application State

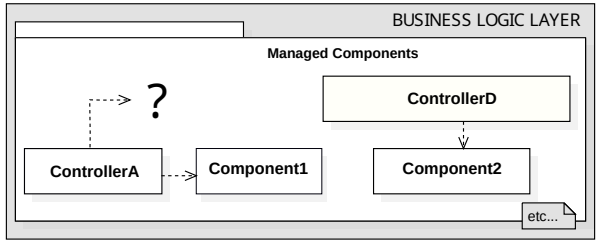
- Memory **purpose**:
  - User session Data
  - Shared Data
  - Internal Information
- **Business Logic** as State Layer:
  - Back-End side
  - **Controllers**: navigation logic
  - **Components**: compound data
- **Transient** state



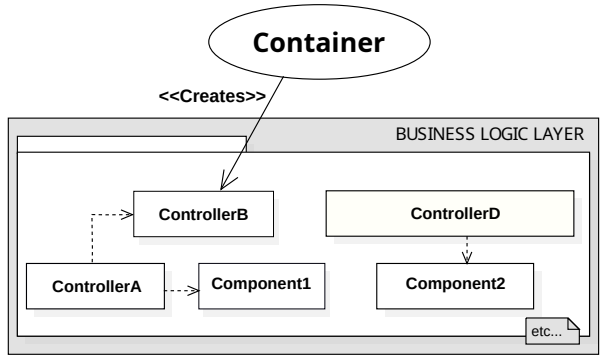
Components Lifecycle Management

- **Container** manages Components Lifecycle
- **Scopes** infer Components Lifecycle:
  - HTTP-based Scopes: *Session* and *Request*
  - Singleton Scope: *Application*
  - Programmatic Scope: *Conversation*
- **Application State**: Aggregation of Components with Different Lifecycles

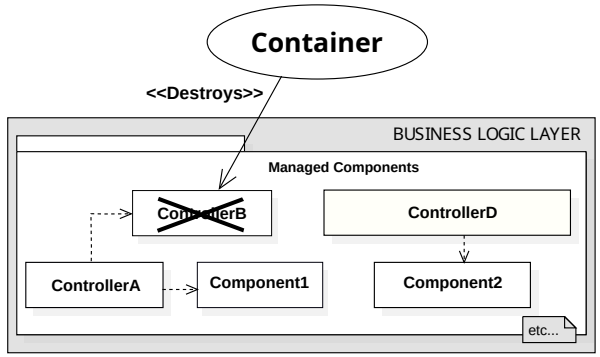
1)



2)



3)



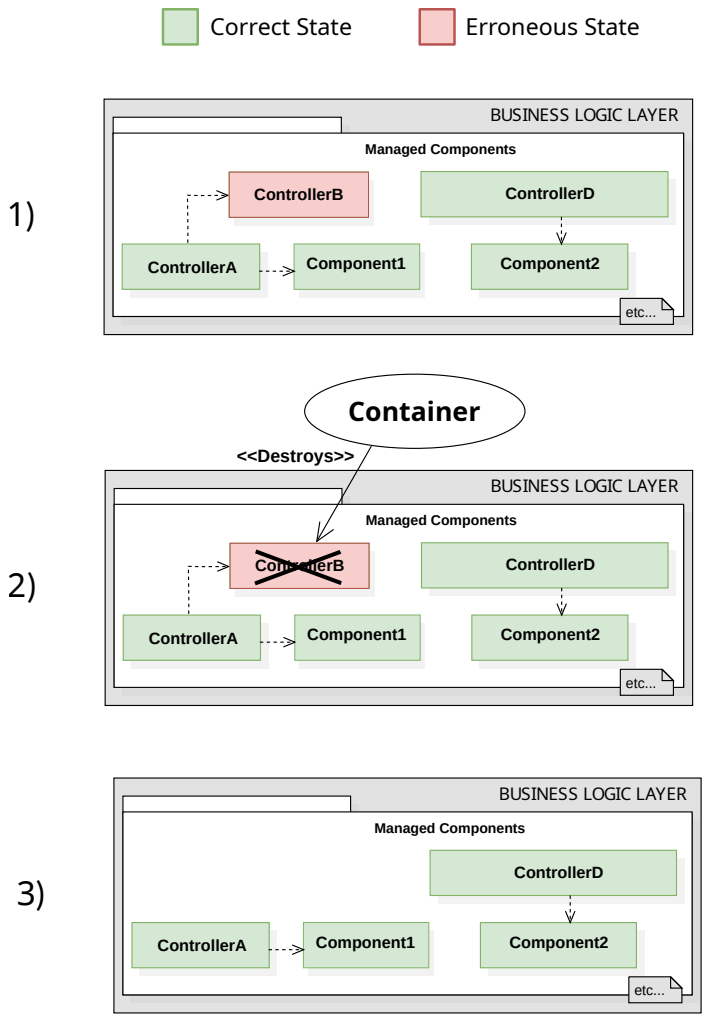
Software Aging in Stateful Web applications

- In-Memory Components could be subject to **software aging**
- Errors could propagate affecting the User Interface
- Errors Activation due to:
  - Source code faults
  - **External faults**
- External faults can not be captured during standard testing phases
- **Focus**: Errors activated by faults arriving over time:
  - Soft Errors [1]
  - Physical Components
  - Application Server overloading
  - Other *ARBs*

[1] Cerveira, Oliveira, Barbosa, Madeira, "Evaluation of RESTful frameworks under soft errors", ISSRE 2020,

Lifecycle Management as Software Rejuvenation

- Destruction and re-instantiation of component as a **safe state restoration**
- Lifecycle management as a kind of SW **micro-rejuvenation**
- Different Lifecycles correspond to **different frequencies** of component restoration

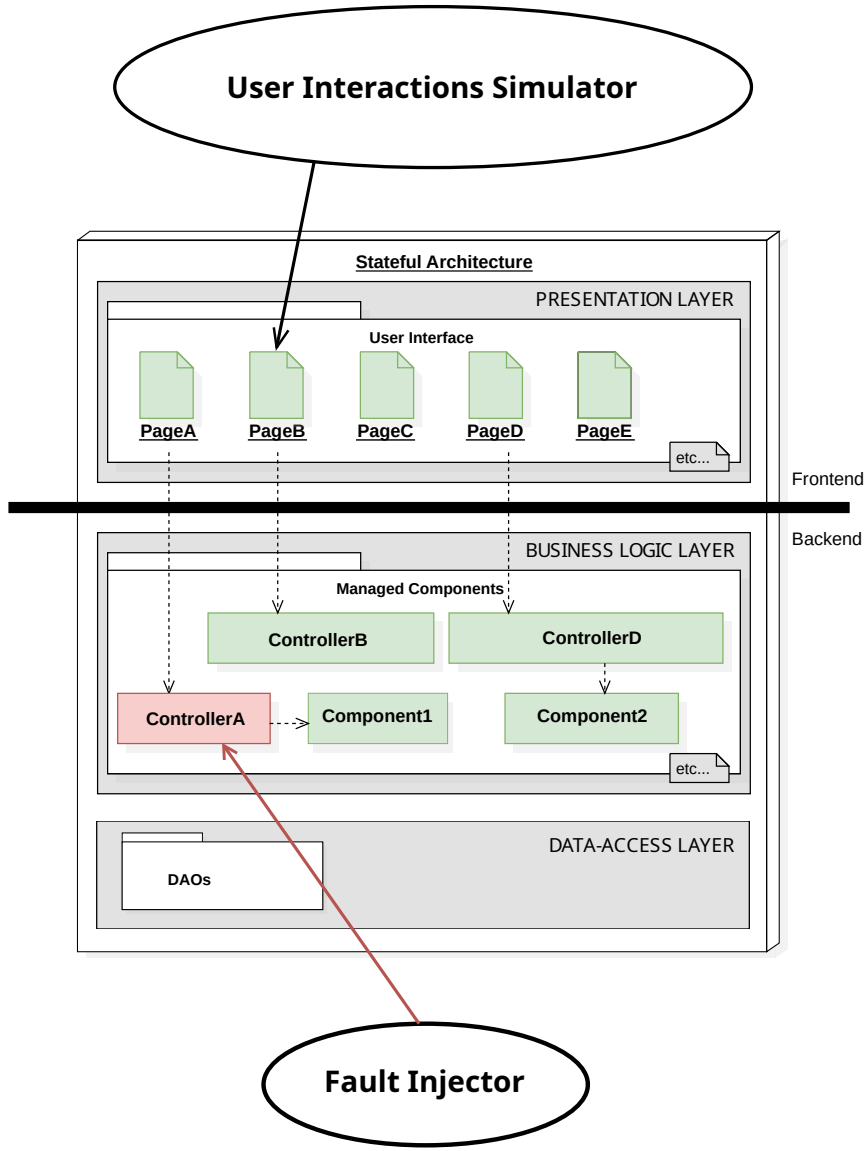


Scope Design Configuration

- **Scope design**: assignment of components scopes
- Different configurations available for the same solution:
  - Session *vs* Conversation
  - Application as in-memory DB
  - Toward the statelessness with Request scope
- Choosing a scope implies a **trade-off**
- Scope design defines a **micro-rejuvenation policy**

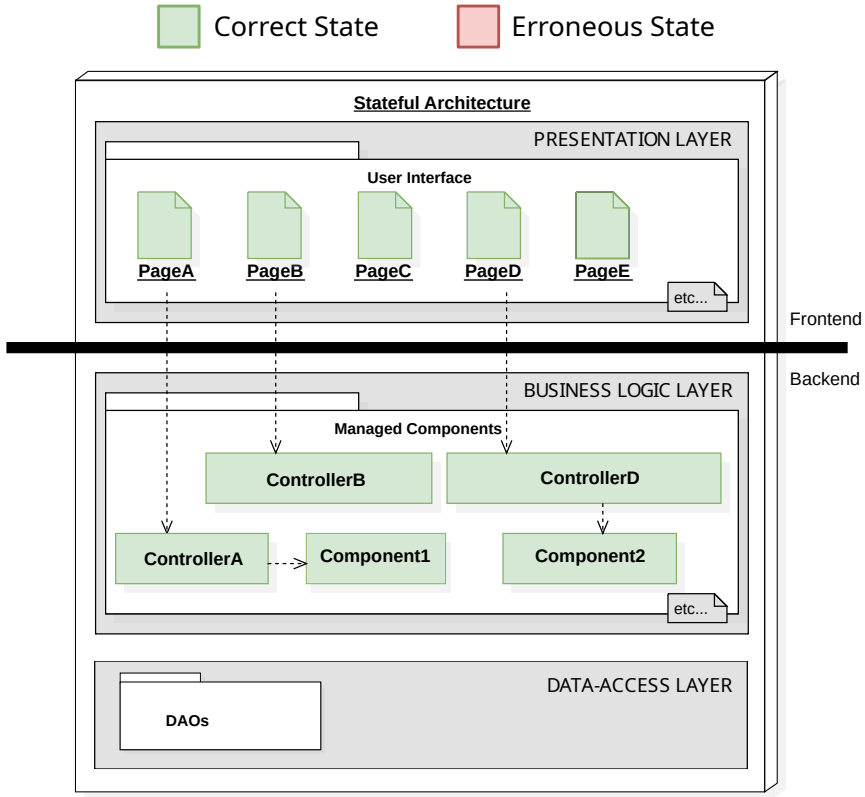
Experimental Environment

- **Goal**:
  - Observe the rejuvenation phenomenon
  - Evaluate the impact of different scope designs
- E2E-like Test Cases **simulating scenarios** where a User interact with the UI
- **Fault Injector** triggered during Scenarios perturb random components
- Developed for JEE with various tools:
  - *Selenium Web Driver*
  - *Arquillian Warp*
  - *Shrinkwrap*
  - *Weld Bean Manager*



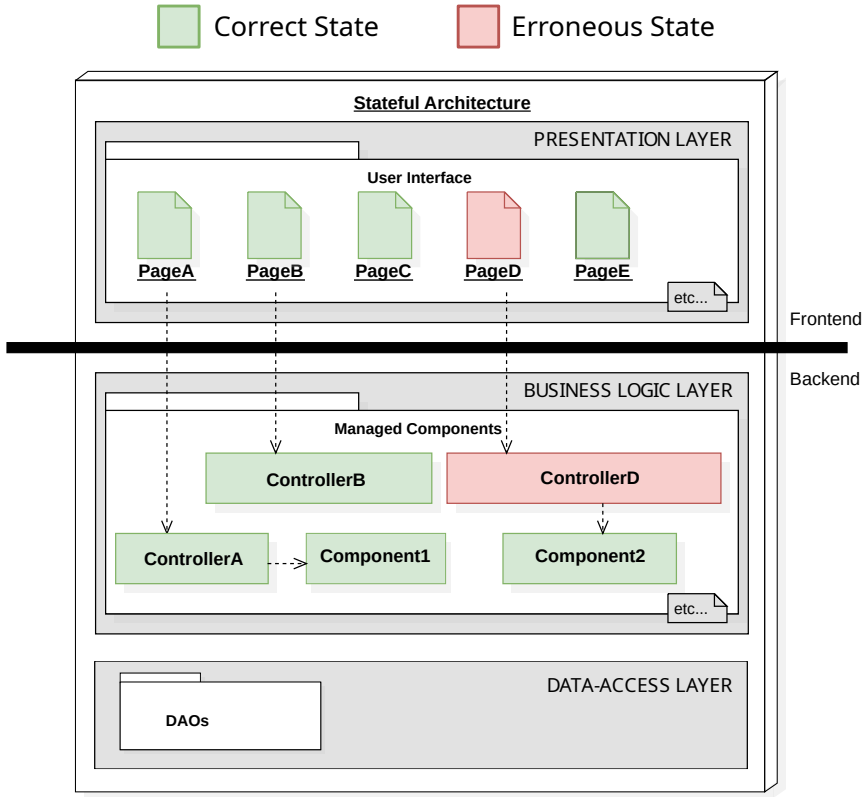
Simulation Scenario Outcomes

- **Error Correction**: the state of the application is correct → *Rejuvenation Succeeded*
- **Manifested Failure**: injected error has caused a top level failure → *Rejuvenation Failed*
- **Latent Errors**: no top level failure has occurred but the application is still in an erroneous state → *Rejuvenation could Fail*



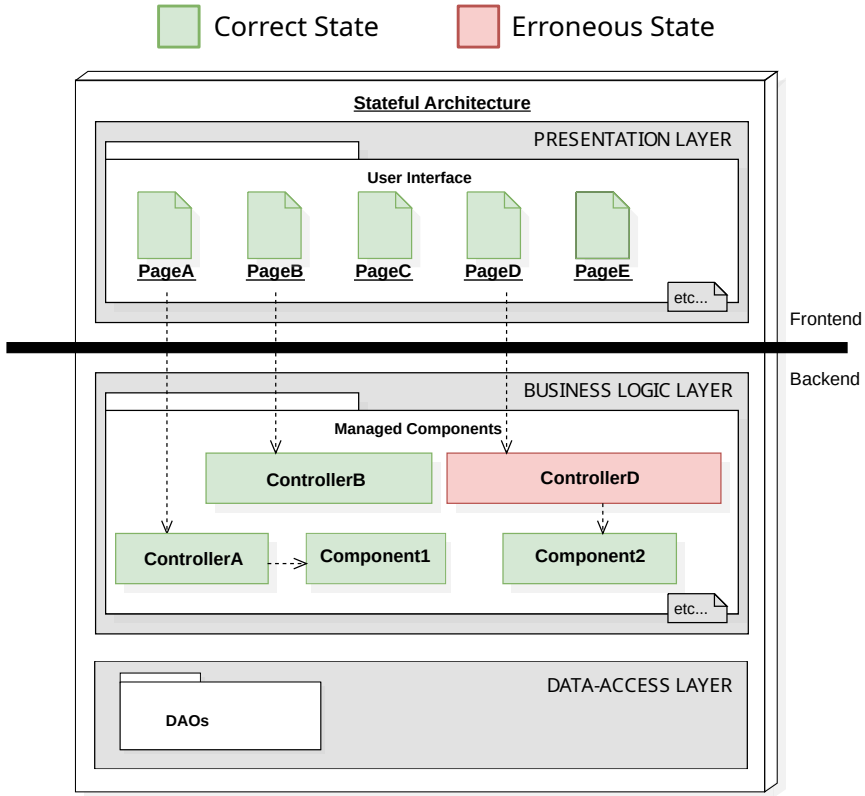
Simulation Scenario Outcomes

- **Error Correction**: the state of the application is correct → *Rejuvenation Succeeded*
- **Manifested Failure**: injected error has caused a top level failure → *Rejuvenation Failed*
- **Latent Errors**: no top level failure has occurred but the application is still in an erroneous state → *Rejuvenation could Fail*



Simulation Scenario Outcomes

- **Error Correction**: the state of the application is correct → *Rejuvenation Succeeded*
- **Manifested Failure**: injected error has caused a top level failure → *Rejuvenation Failed*
- **Latent Errors**: no top level failure has occurred but the application is still in an erroneous state → *Rejuvenation could Fail*



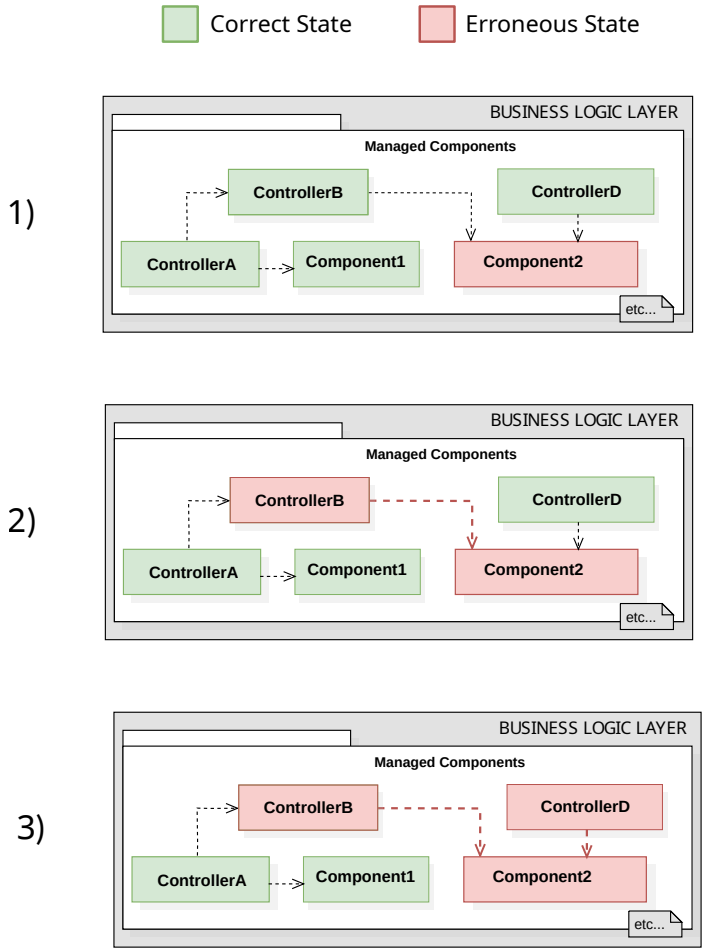
Scope-Wise Experimentation Results

- No Correction capability for Application Scoped components
- **Wider Scopes**: high failure and error latency rates
- **Lower Scopes**: high error correction rate

Scope	Manifested failures (%)	Latent errors (%)	Corrected errors (%)
application	**42.4**	57.6	**0**
session	**48**	36	16
conversation	12	24	64

Error Propagation

- Error could be propagated during **components interactions**
- Error Accumulation problem
- **Increase** the Failure capability
- **Decrease** the Correction capability



Error Propagation Evaluation

- Error propagation does **not only depends** on lifecycle
- **Dependencies** have an impact on propagation
- **Interactions** is a core aspect
- Session Components have the best combination of both

Scope	Errors propagation (%)	Mean propagated errors	Mean touched components	Prop. ratio (%)
application	44	**1**	**4.6**	21.7
session	**32**	**1.33**	2.5	53.2
conversation	16	1.25	1.7	73.5

Policy-Wise Experimentation Results

- Two **functionally equivalent** versions of the same application with different scopes design policies
- **Data Long Retention**: higher failures and error latency rates
- **Lower Scope**: higher error correction rate
- Consistent with the Scope-Wise experimentation

Principle	**Manifested failures** (%)	**Latent errors** (%)	**Corrected errors** (%)
data long retention	42	46	12
lower scope	30	26	48

Discussion on Future Work

- Evaluate also the case of faults arriving over **sequence of events**
  - Could be done easily with the same Experimental environment
- Characterise the **trade-off** involved when a component is assigned to a specific scope
  - Usually assigning a scope implies a trade-off between safety and performance
- Identify a methodology to guide the scope design phase **tailored** to the application under development
  - An optimised components lifecycle assignment
  - Aware of **dependencies and interactions**