Software Patterns for Runtime Variability in Online Enterprise Software

Jaap Kabbedijk & Slinger Jansen

BENEVOL 2013

Utrecht University

December 16, 2013

Outline

What is it about?

Software Patterns for **Runtime Variability** in **Online Enterprise Software**

We need some Q&A!

Q: Online Enterprise Software?

A:

- Enterprise software is increasingly moving towards the cloud [DKS+12]
  - Rapid deployment
  - Increased product innovation
  - Reduced costs
- Makes increasing use of Multi-tenancy [BZ10]
  - Serving multiple tenants from one application
  - Varying customers
  - Sharing resources

Q: Runtime Variability?

A:

- One code base
- Different customers have different wishes
- The system should support tenant-specific requirements
- Should be able to dynamically adapt functionality [SVGB05]
- Ideally, a software product 'evolves', or changes, according to tenant-specific requirements

Q: Software Patterns?

A:

- General solution to a recurring problem
- Present a proven **idea**, no **implementation**
- Often include **consequences** [KJ12]

Q: So, what is the problem?

A:

- Unclear how to implement variability
  - Functional level
  - Data level
- Unclear what are best fitting or appropriate solutions, based on the context

Research Approach

- Design Science approach [HMPR04]
- Multiple case studies
  - All current commercial products
  - One of the authors took part as consultant
- Evaluation by domain experts
  - First part: Semi structured interview
  - Second part: Free discussion on quality attributes

Outline

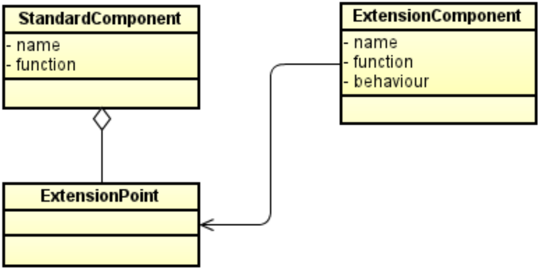
Introduction

Dynamic Function Adaptation (DFA) Patterns

- DFA: Component Interceptor Pattern
- DFA: Event Distribution Pattern

Wrap-up

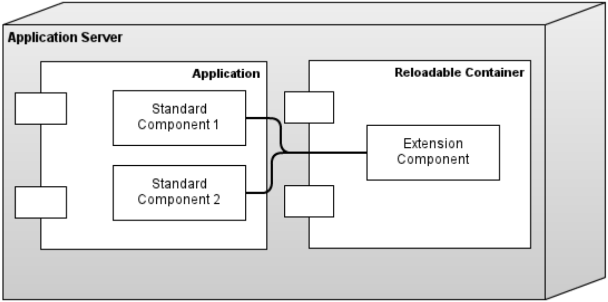
DFA: Problem Statement



Example: Sending a notification to transportation department if tomorrow's batch will be bigger than normal

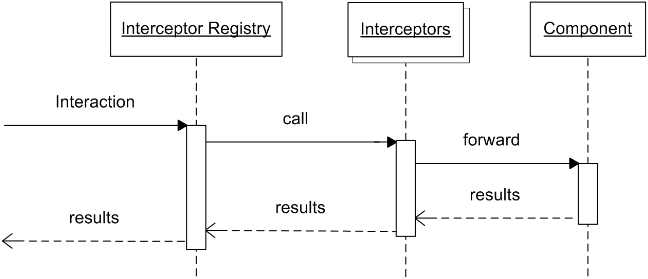
Outline

Component Interceptor Pattern: System Model



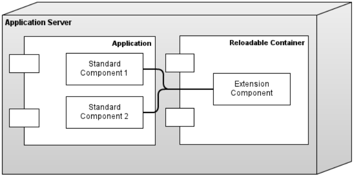
Example: OSGi for dynamically reloading code (reloadable container) in Java

Component Interceptor Pattern: Sequence Diagram



Note: System can not continue until all interceptors in registry finished executing

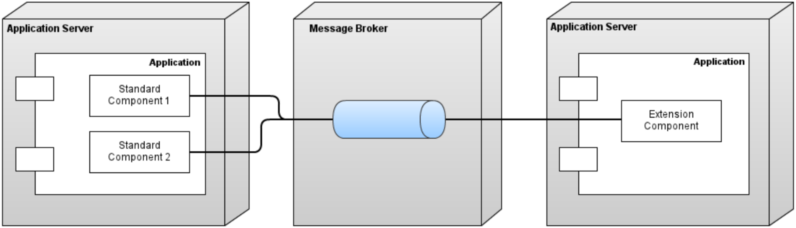
Component Interceptor Pattern: Characteristics



- Single application server
- Interceptors run in-line with normal code
- Access to all arguments
- Able to modify all argument and data

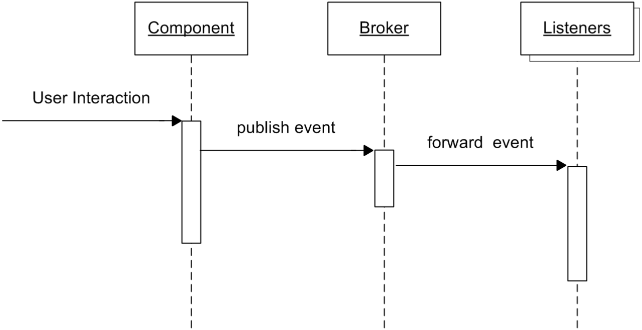
Outline

Event Distribution Pattern: System Model



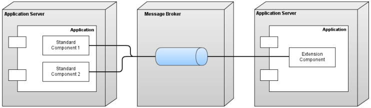
Example: JBoss Messaging for handling requests (message broker) in Java

Event Distribution Pattern: Sequence Diagram



Note: System does not know if an event happened. Waiting and/or rollbacks are options

Event Distribution Pattern: Characteristics



- Distributed nature
- Listeners are loosely coupled
- Access through API
- Components unaware of listeners

Which solution is best?

Which solution is best?

Let's look at some Quality Attributes...

Security - Performance - Scalability - Maintainability - Impl. Effort

Adding functionality always adds potential security threats

- Component Interceptor Pattern
  - Extension components have full access
  - Extension components are not isolated
- Event Distribution Pattern
  - Extension components are isolated
  - Extension components communicate through an API

Security - **Performance** - Scalability - Maintainability - Impl. Effort

- Component Interceptor Pattern
  - Extension components are part of the system
  - No need for (un)marshalling
- Event Distribution Pattern
  - Distributed
  - Extra network resources
  - (un)marshalling

Security - Performance - **Scalability** - Maintainability - Impl. Effort

- Component Interceptor Pattern
  - One application server
  - Scaling up is difficult
  - Interceptors must be known to all servers
- Event Distribution Pattern
  - Distributed
  - Easy to add extra servers

- Performance - Scalability - **Maintainability** - Impl. Effort

More variability always causes more testing and more extensive maintenance

- Component Interceptor Pattern
  - Changing parameters will directly influence extension components
- Event Distribution Pattern
  - Changing parameters does not directly influence extension components, because of API

Security - Performance - Scalability - Maintainability - **Implementation Effort**

Both patterns need extension points

- Component Interceptor Pattern
  - Interceptor Registry
  - Normal function calls
- Event Distribution Pattern
  - Message Broker
  - API calls

P1 & P2: Comparison

- Component Interceptor Pattern (P1) → For small projects
  - Good performance on one application server
  - Low implementation effort
- Event Distribution Pattern (P2) → For large project
  - Secure
  - Scalable
  - Easy to maintain if a project gets larger

Outline

Introduction

Dynamic Function Adaptation (DFA) Patterns
- DFA: Component Interceptor Pattern
- DFA: Event Distribution Pattern

Wrap-up

Future Work

- Identify more domains for variability problems
- Identify more solutions
- Perform more comparisons
- Evaluation of the **solution**, instead of the **implementation**

What to take home?

- Patterns are helpful for tackling variability problems
- Comparison of similar patterns is crucial
- This work is never done... or is it?

How can you help?

We are planning on doing something similar for Dynamic Data-model Adaption (DDA) patterns.

- Identified two
- Compared both

What should we potentially adapt based on current work?

Questions

j.kabbedijk@uu.nl
www.jkabbedijk.nl

References I

C.P. Bezemer and A. Zaidman.
Multi-tenant SaaS applications: maintenance dream or nightmare?
In *Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE)*, pages 88–92. ACM, 2010.

Austin D'souza, Jaap Kabbedijk, DongBack Seo, Slinger Jansen, and Sjaak Brinkkemper.
Software-as-a-service: Implications for business and technology in product software companies.
In *Proceedings of the Pacific Asia Conference on Information Systems (PACIS)*, Paper 140, 2012.

Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram.
Design Science in Information Systems Research.
MIS Quarterly, 28(1):75 – 105, 2004.

References II

Jaap Kabbedijk and Slinger Jansen.
The role of variability patterns in multi-tenant business software.
In *Proceedings of the WICSA/ECSA 2012 Companion Volume*, pages 143–146. ACM, 2012.

Jaap Kabbedijk, Tomas Salfischberger, and Slinger Jansen.
Comparing two architectural patterns for dynamically adapting functionality in online software products - best paper award.
In *Proceedings of the 5th International Conferences on Pervasive Patterns and Applications (PATTERNS 2013)*, pages 20–25, 2013.

Mikael Svahnberg, Jilles Van Gurp, and Jan Bosch.
A taxonomy of variability realization techniques.
Software: Practice and Experience, 35(8):705–754, 2005.

Final Notes

- Initial results published at PATTERNS2013 [KSJ13]
- Final results to be submitted as (invited) journal publication
- Copyright to all images used in this presentation belongs to their original copyright holders