Universiteit Utrecht

[Faculty of Science
Information and Computing Sciences]

Variability in Multi-tenant Environments:

Architectural Design Patterns from Industry

Jaap Kabbedijk, MSc.
Dr. Slinger Jansen

Utrecht University

Outline

- Introduction
- What does everything mean?
- Research Approach
- Example Design Patterns
- Conclusion

Introduction

- Software is increasingly offered 'as a Service' (Ma, 2007)
- Customers want a product to do 'what they want'
- Complying to customer specific requirements in a SaaS environment has drawbacks:
  - Difficulty with scalability
  - Difficulty with maintainability
  - Architectural erosion
- Multi-tenancy enables software vendors to offer a SaaS product in a cost efficient way (Guo et al., 2007)

Problem Statement

- Software vendors have problems implementing variability in their online products

- We will offer design patterns that help software vendors in implement variability in multi-tenant SaaS products

What is Multi-Tenancy? (1/2)

The ability to let different tenants share the same hardware resources, by offering them one shared application and database instance, while allowing them to configure the application to fit their needs as if it runs on a dedicated environment (Bezemer & Zaidman, 2010)

What is Multi-Tenancy? (2/2)

- Data Model Multi-tenancy

  One database (or multiple databases balanced with the same data model) is shared by all customers

- Application Multi-tenancy

  One instance (including database) of the software product is used to serve all customers

- Full Multi-tenancy

  Customers are all served by one single database and instance of the software product, while being able to have the functionality they want

What is Variability?

- The ability of a software system or artefact to be efficiently extended, changed, customized or configured for use in a particular context (Svahnberg, van Gurp & Bosch, 2005).
- Within a SaaS product it is difficult to offer a different variant to all customers without solid design patterns enabling this.

Variability moments

- During design
  - Different product for Linux than for Windows
- During compilation
  - Point to different sections of code while compiling software for a specific phone
- Linking at installation
  - Linking a product to several additional modules
- **Run-time**
  - When a user of an on-line system wants to change something

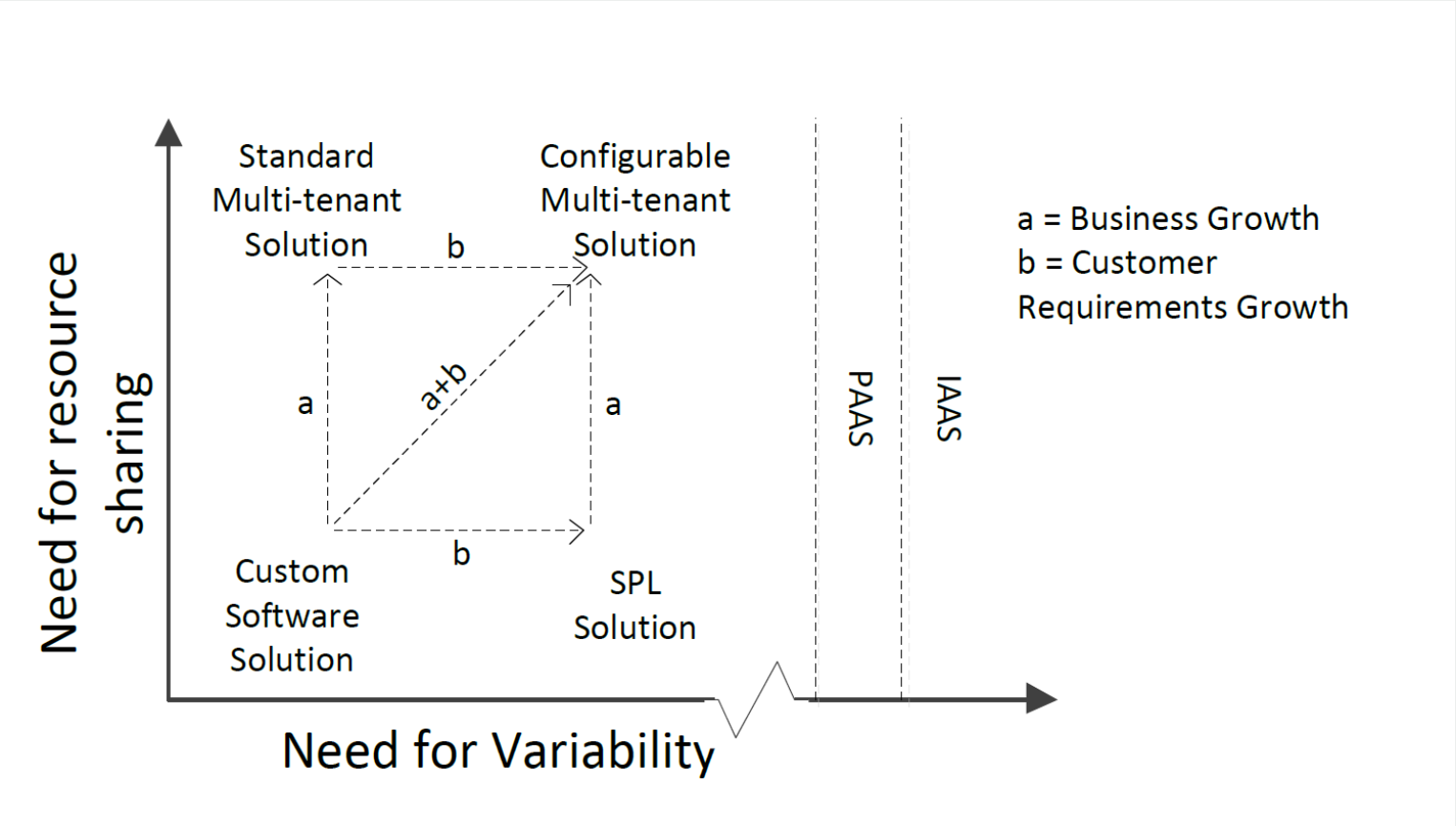
What are Design Patterns?

A pattern for software architecture describes a particular recurring design problem that arises in specific design contexts, and presents a well-proven generic scheme for its solution. The solution scheme is specified by describing its constituent components, their responsibilities and relationships, and the ways in which they collaborate (Buschmann et al., 1996)

Research Method

- Case Study Research
  - Exact (ErpCompA)
    - 20.000 users
    - 7 different feature modules
  - AFAS (ErpCompB)
    - 10.000 users
    - 9 different feature modules
- Literature review
  - 27 papers
    - Variability AND SaaS
    - Variability AND multi-tenancy

When do you need Multi-tenancy?

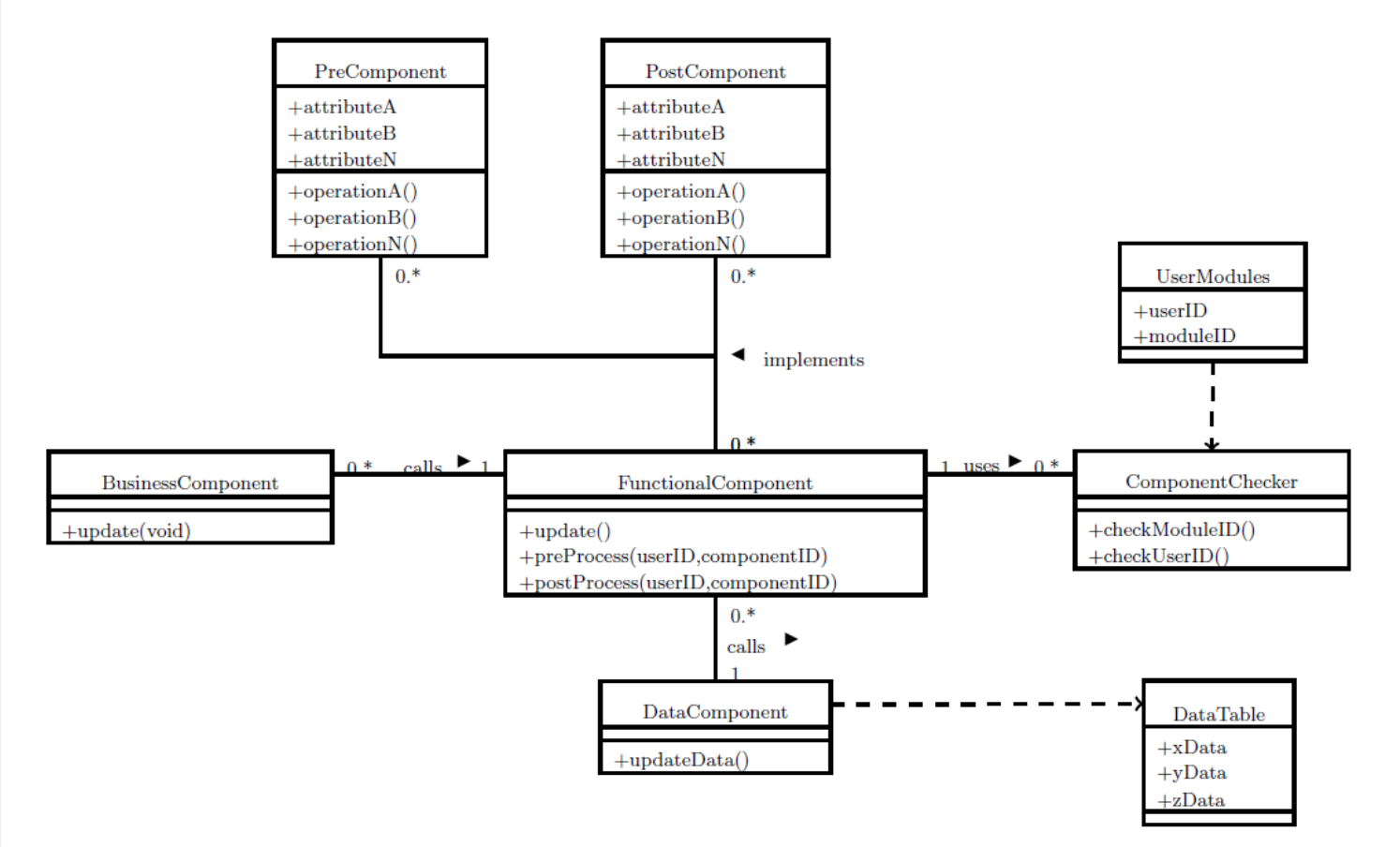


Pattern Development

- Identify Run-time Variability Patterns currently used
  - Case study at AFAS
  - Case study at Exact
- Identify RVPs in literature (literature study)
- Come up with new RVPs (design science)

Example Patterns

Pre/Post Update Hooks (1/3)



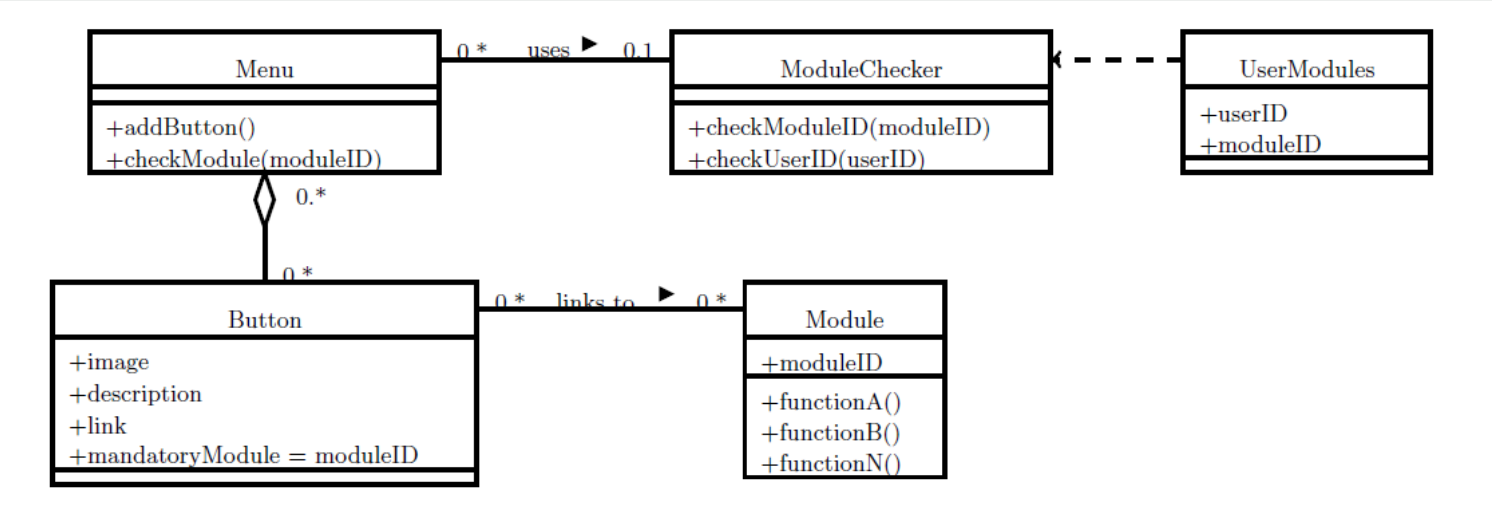
Pre/Post Update Hooks (2/3)

- Intent - To provide the possibility for tenants to have custom functionality just before or after an event
- Motivation - To let the software product fit the tenants business processes best, extra actions could be made available to tenants before or after an event is called
- Solution – The use of a component able of calling other components before and after the update of data. The tenant-specific modules are listed in a separate table

Pre/Post Update Hooks (3/3)

- Explanation – See image
- Consequences - Extra optional components have to be available in the software system in order to be able to implement this pattern
- Example - In a bookkeeping program, tenants can choose, whether they want to update a third party service as well by using a component that uses the API of a third party service to make changes there. If so, the FunctionalComponent can call the third party communicator after an internal update is requested

Module Dependent Menu (1/3)



Module Dependent Menu (2/3)

- Intent - To provide a custom menu to all tenants, only containing links to the functionality relevant to the tenant
- Motivation - Displaying all possible functionality in the menu would decrease the user experience of tenants, so menus have to display only the functionality that is relevant to the tenant
- Solution – Every time a tenant displays the menu, the menu is built dynamically based on the modules he has selected or bought

Module Dependent Menu (3/3)

- Explanation – See image
- Consequences - To be able to use this pattern, an extra table containing user-IDs and the modules available to this user has to be implemented. Also, the extra class ModuleChecker has to be implemented
- Example - In a large bookkeeping product, containing several modules that can be bought by a tenant, the menus presented to the tenant can be dynamically composed based on the tenant's license

Conclusion and Further Research

- Design patterns are an appropriate way to document and communicate solutions to the variability problem
- Are they the best way?
- The concept on multi-tenancy should be better defined
- The proposed patterns should be evaluated
- How are patterns really used?

Discussion

Kabbedijk, J., Jansen, S. (2011). Variability in Multi-tenant Environments: Architectural Design Patterns from Industry. Lecture Notes in Computer Science, 6999, 151-160

www.jkabbedijk.nl

www.productasaservice.org