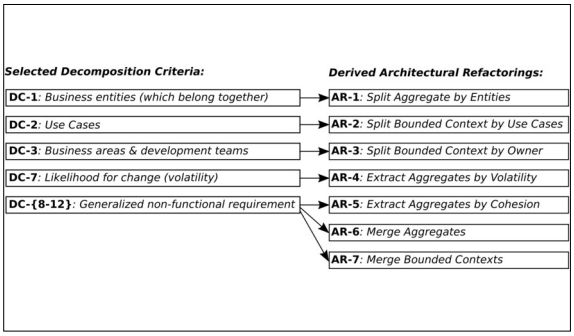
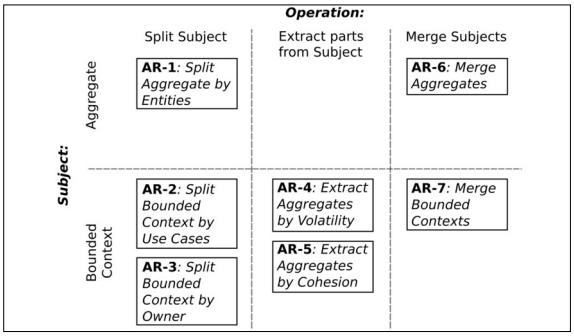
Stefan
Kapferer

Student	Stefan Kapferer
Examiner	Prof. Dr. Olaf Zimmermann
Subject Area	Software and Systems

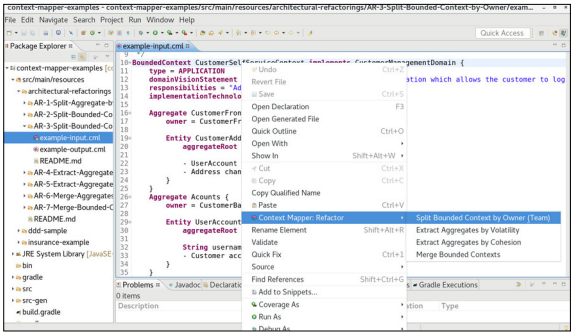
Service Decomposition as a Series of Architectural Refactorings



Architectural Refactorings (ARs) from Decomposition Criteria (DC) compiled from literature and own experience.
Own presentment



The ARs allow to decompose (split and extract) and compose (merge) the DDD bounded contexts and aggregates.
Own presentment



The ARs have been implemented as Code Refactorings for the Context Mapper DSL (CML) and integrated into the Eclipse IDE.
Own presentment

Introduction: Decomposing a system into modules or services always has been a hard design problem. With the current trend towards microservices, this issue has become even more relevant and challenging. Domain-driven Design (DDD) with its Bounded Contexts provides one popular technique to decompose a domain into multiple parts. The open source tool Context Mapper, developed in our previous term project, offers a Domain-specific Language (DSL) for the strategic DDD patterns. DSL and supporting tools assist architects in the process of finding service decompositions. Context Mapper has already been used in practice projects, which led to suggestions how to improve the DSL to further increase its usability. Moreover, Context Mapper at present does not offer any transformations or refactoring tools to improve and evolve the DDD models. Finally, our previous work only gives very basic advice on how to implement systems that have been modeled in Context Mapper in a (micro-)service-oriented architectural style.

Result: This work presents a series of Architectural Refactorings (ARs) for strategic DDD models based on corresponding Decoupling Criteria (DC) collected from literature and personal experience. These refactorings allow a software architect to (de-)compose a domain iteratively. Aiming for a broad DC coverage, a set of seven ARs has been implemented. These ARs are realized as code refactorings for the Context Mapper DSL (CML) language and support splitting, extracting and merging Bounded Contexts and/or Aggregates. Therefore, DSL users are able to refactor their CML models within the provided Eclipse plugin. A new service contract generator offers assistance how to implement the DDD models in an (micro-)service-oriented architecture. The resulting contracts are written in the Microservices Domain Specific Language (MDSL), another emerging DSL for specifying service contracts.

Conclusion: The provided DSL with its seven ARs, implemented as model transformations, support evolving DDD-based models in an iterative way. The conducted validation activities support our hypothesis that software architects can benefit from such an approach and tool. Action research has been applied to improve Context Mapper in each iteration of the prototypical implementation. Basic case studies conducted on real world projects in the industry indicated the usefulness and effectiveness of the modeling language. More advanced validation activities still have to be conducted to analyze and demonstrate the practicability of the ARs.