9 Research Questions Resolved

"All truths are easy to understand once they are discovered; the point is to discover them." —Galileo Galilei.

9.1 Main Contributions

In section 1.6 the thesis introduced the research questions that were defined in the beginning of the research activities. The central research focus driving my work was the question: Do patterns provide a useful abstraction and adequate means to better understand and analyze existing software architecture?

In summary, this thesis proves that patterns provide an effective tool for describing and understanding single applications, platforms, application domains, and even paradigms such as service-orientation. I’ve used the domain of remoting middleware as example throughout my research activities. Thus, the core contribution of this thesis is the provisioning of a pattern language for the domain for remoting middleware which even proves to be useful for explaining paradigms such as Service-Oriented Architecture (SOA) and further extending paradigms such as SOA and Distributed Object Computing. The pattern language provided is also applicable to derive best practice patterns useful to leverage remoting middleware and SOA technologies in an efficient and effective way. The domain of remoting middleware is only used as an example domain. Hence, other domains are also expressible using a pattern-based approach.

In detail, the thesis has resolved the research topics as follows:
Chapter 9 - Research Questions Resolved

[Q1]: Can we provide a scientific description and classification of software patterns that includes the usage of patterns for architectural analysis?

Answer: In section 2.1 this thesis offers a new systematic description of patterns as a general software engineering discipline as well as a classification and categorization. A pattern mining process helps to evaluate and analyze existing software architectures for extracting patterns and proto patterns. In addition, pattern mining was introduced as a means for the systematic derivation of best practice patterns.

[Q2]: In order to understand the key concepts of distribution middleware: can we systematically classify and describe middleware in general and remoting middleware in particular as well as Service-Oriented Architecture in terms of requirements (commonalities) and concepts?

Answer: Section 2.2 systematically classifies and describes the domain of distribution middleware and its properties. The thesis applies a kind of commonality analysis to identify the most important requirements a remoting middleware technology must meet, starting with fundamental requirements, over advanced requirements, up to additional services. More specific requirements for Service-Oriented Architecture are elaborated in chapters 4 and 5. This served as groundwork for the thesis.

[Q3]: Is it possible to introduce new or use existing patterns and proto patterns to describe the relevant core concepts, e.g., can remoting middleware being described with patterns?

Answer: The thesis could identify and document a system of patterns providing solutions for the fundamental design problems arising in the domain of remoting middleware in chapters 3, 4, 5, 6. These patterns have been applied in all existing remoting middleware solutions and thus express the core concepts of remoting middleware in general. For example, the Broker pattern describes the general principle underlying all those remoting middleware solutions.
Chapter 9 - Research Questions Resolved

[Q3]b: Can software patterns even help to describe paradigms such as remote objects or service-oriented architecture?

Answer: As already pointed out in the answer to research question [Q3]a, the thesis could prove that all relevant remoting middleware solutions in principle use the same patterns in possibly different variants. In chapter 5, the thesis could also approve that paradigms are expressible using patterns. As a paradigm such as SOA addresses different problems that could not be solved fully at least not efficiently with previous approaches, the solutions to these problems are (proto) patterns themselves.

[Q3]c: Are software patterns applicable to describe and compare the different remoting middleware technologies?

Answer: Chapter 3 introduces a system of patterns that describes the domain of remoting middleware and provides a means to idiomatically express commonalities (i.e., the patterns) as well as variabilities (i.e., patterns variants or additional functionalities) between various concrete middleware technologies.

[Q3]d: How can software patterns be evolved and refined with more details using additional software patterns to further cover the domain of remoting middleware?

Answer: Chapter 6 introduces the Broker Revisited pattern that refines and modifies the original Broker pattern. This way, the thesis could illustrate that further refinement or modification of patterns also leads to further refinement and modification of domain and paradigm concepts. The reason for this is obvious. If an application, domain or paradigm is expressed using patterns, then the pattern systems reveal possibilities for systematic modification, refinement or extension. For example, chapter 5 not only presents the core concepts of the SOA paradigm using patterns. It also helps to identify possible future evolution directions. Chapters 7 and 8 introduce patterns for the further detailing and refining of remoting middleware systems. These chapters demonstrate that for architectural analysis and understanding of any given software system a coarse-grained coverage
with patterns is sufficient, but that more and more details can be uncovered using additional patterns to describe lower abstraction levels. Basically, this means that a top-down approach is essential in this context which is also shown by the pattern mining process introduced.

**[Q3]** Can best practice patterns be derived that help application programmers leverage the remoting middleware?

Answer: As presented in section 2.1.14 as well as in chapter 5, the pattern systems used to describe (families of) software architectures support the derivation of best practice patterns. The documented consequences of the pattern application as well as properties of the underlying system infrastructure provide input to define best practices.

**[Q4]** Does architectural coverage of a domain or software system necessarily imply complete coverage by a pattern language or is it sufficient to express only the core concepts using patterns?

Answer: The thesis could prove that for domains such as remoting middleware it is not necessary to provide full pattern coverage if the central concepts of the domain can be expressed using patterns. As core concepts and core requirements of a domain denote recurring problems within that domain, it is obvious that there are also common solutions. Areas for which no patterns exist can be expressed using proto patterns. Since pattern description templates prove to be an efficient and effective means for describing, understanding and documenting all kinds of architectural entities, any existing software architecture is expressible using systems of patterns and proto patterns documented by pattern description templates. At least, patterns and proto patterns should cover the core abstractions at higher architecture levels. This is possible because core concepts solve recurring problems using appropriate solutions. For a complete domain, these problems must be addressed by multiple applications so that the concepts are idiomatically expressible using patterns.
9.2 Possible Future Research Topics

“Somewhere, something incredible is waiting to be known.” —Carl Sagan.

There are various levels of topics that could be subject to further investigation. Note, that some of these topics are not closely related to the research subject of this thesis, but rather define more general and relevant challenges for software architecture and middleware research.

- How could tool support for pattern mining look like? Currently, architecture analysis tools such as Sotograph focus more on the level of language-specific aspects like classes, packages. They allow the architect to specify sub-system organization. It is very unlikely that automatic tool support for detecting patterns will work as patterns denote blueprints with potentially infinite implementations. Thus, human interaction and guidance will be necessary.

- I see a demand for pattern systems that deal with particular operational and developmental requirements. In addition, further research on systematic injection of these properties into software architecture is required. For this purpose, all relevant operational and developmental properties must be specified in detail. For qualities that represent cross cutting concerns, aspect-oriented software development could be a good means.

- What about using pattern systems that cover domains such as remoting middleware as fundamental core of Model-Driven Software Development tools which generate distributed applications?

- If a domain can be partitioned into sub-domains and if for all of these sub-domains pattern languages exist, is it then possible to compose these pattern languages to a pattern language of the whole domain? In other words, how can understanding sub-domains help to understand the target domain? A similar question: given the
existence of DSLs for sub-domains, is it possible to derive a DSL for the whole domain?

- As pointed out in 2.1.15 software patterns might help to introduce refactoring for all abstraction levels of the software architecture. Even the difference between re-engineering and refactoring might be eliminated. In a further step pattern systems might help steer refactoring activities.

- How can feature modeling and patterns be used in conjunction for architecture recovery and architecture construction?

- In pattern-based systems, software metrics such as cyclomatic complexity lead to inappropriate results as these metrics do not take patterns into account. What is the right way to address this problem for metrics and software analysis tools?

- This thesis focused on remoting middleware and SOA as examples. Similar analyses could be done for other types of middleware. And of course, similar investigations could also be done for other domains.

- Are there further types of patterns for other abstraction layers? As an example look at patterns such as Reactor, Interceptor, or Observer. All these patterns look quite similar, providing registration as well as notification functionality. Are there common core patterns that are included in these patterns and can thus be obtained from patterns such as Reactor, Interceptor, and Observer?

- Can paradigms like OOP (Object-Oriented Programming) be expressed idiomatically using patterns? In my opinion, the answer is yes, as for instance OOP defines common solutions for recurring problems such as the concept of classes, interfaces, and objects.