NEXOF-RA

NESSI Open Framework – Reference Architecture

IST- FP7-216446

Deliverable D7.2c

Definition of an architectural framework & principles

Piero Corte
Debora Desideri
Ricardo Jimenez-Peris
Francisco Perez Sorrosal

Due date of deliverable: 30/11/2010

Actual submission date: 30/11/2010

This work is partially funded by EU under the grant of IST-FP7-216446.

Change History

sion	Date	Status	Author (Partner)	Description
	10/11/2009	Draft	Debora Desideri, Piero Corte	First draft
	19/11/2009	Draft	Debora Desideri, Piero Corte	**1 SECTION THE SPECIFICATION PROCESS** The *NEXOF-RA Project* has adopted a top-down production process to b support the production of a set of inter-related patterns. Starting from production of top-level patterns, i.e. the most general and abstract patt other patterns are produced with respect to other already-committed patt This way, the problem they address and the context where they are applic are clearly and well-defined. This approach makes the verification of consistency of the overall set of patterns as well as the instantiation of con architectures easier and more controllable. The next figure gives a very high-level description of the pattern develop process.

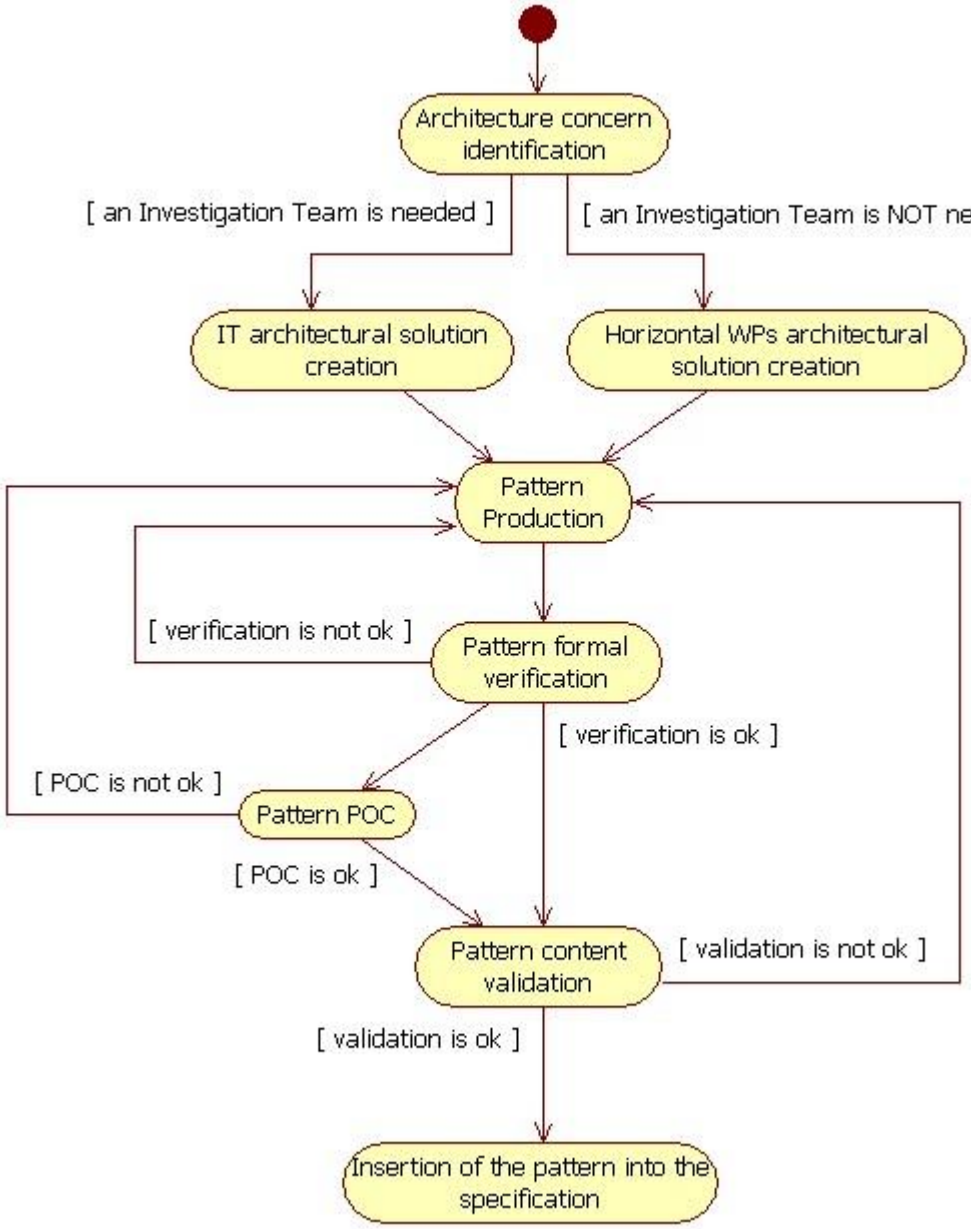


Figure 13: Patterns Development Process

When an architectural issue, related to SOA infrastructures, is identified *NEXOF-RA Project* expert people start working on the elaboration of architectural patterns. This activity is performed within the research trac the Project and can be supported by means of external contributors Investigation Teams). Contributors must use the pattern template docume produce the pattern description document.

After new patterns have been produced, they are submitted to the specific development workgroup (WP7) to check their relationships to the other pat of the specification, the compliancy to the pattern template and the consist of functionality requirements and assumptions with respect to the s

relationships. This checking action is called “formal verification” though it i based on any formal logic. It mainly deals with constraints on the format c pattern description and takes it apart from the content validation that evalu the quality of the pattern’s architectural choices. If the formal verification i ok, WP7’s feedback is returned to team working on the elaboration o pattern, where an update version of the pattern will be elaborated. This c can be repeated several times till the pattern succeeds WP7 verification.

Once WP7 verification is achieved, those patterns that address challenging architectural problems of SOA Infrastructure design are selected to be valid through proof-of-concept actions. In any case all the patterns are submitted the *NEXOF-RA Architectural Board* (AB) for the validation of the content. In phase, the pattern is evaluated with respect to its architectural design adde value, originality and innovation. If it is validated by the architectural board, pattern will be added to the *NEXOF-RA Specification*, otherwise the pattern the negative feedbacks will be returned back to the development team for further improvements.

To better explain the above process, next figure depicts all the various state pattern can be during its development process.

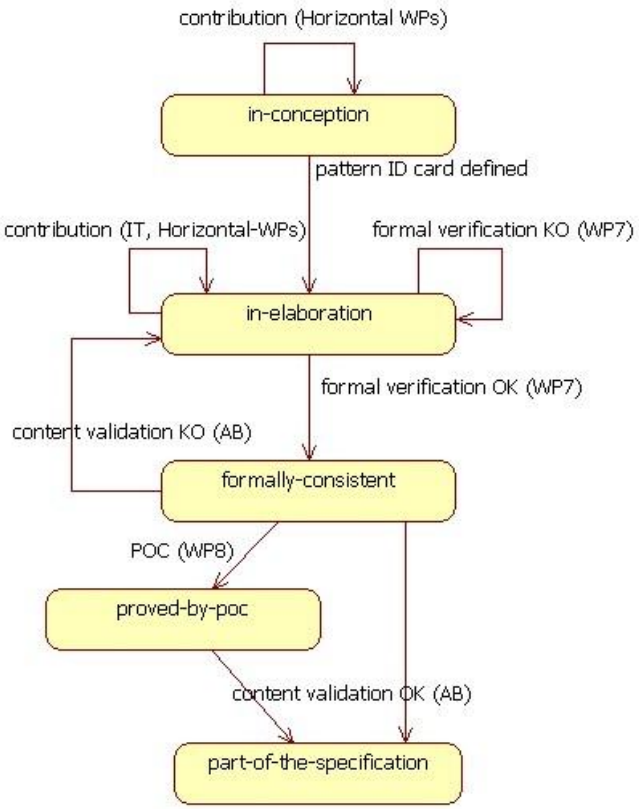


Figure 14: Pattern States

As shown by the previous figure, a pattern can be in one of the following sta

- **in-conception**: to refer to a pattern that has been identified and qua

				only by means of the architectural problem it addresses. The architec solution proposed by a pattern in this state has not been elaborated yet. • **in-elaboration**: to refer to a pattern that is under development. A draft c architectural solution proposed by a pattern in this state is available. • **formally-consistent**: to refer to a pattern that has been already elabor and has succeeded the formal verification. • **proved-by-poc**: to refer to a pattern that has been successfully proved proof-of-concept action. • **part-of-the-specification**: to refer to a pattern that has succeeded formal verification and has received the final approbation of the *NEXO Architectural Board* to be part of the *NEXOF-RA Specification*. modified to add POC in the process
	23/11/2009	Draft	Debora Desideri, Piero Corte	Section Relationships to Components Catalogue, Relationships to Standards Cata changed
	23/11/2009	Draft	Debora Desideri	Minor changes
	23/11/2009	Draft	Piero Corte	Document review
	30/11/2009		Debora Desideri, Piero Corte	Review according Evelyn comments
	30/11/2009	Draft	Debora Desideri, Piero Corte	Appendix A removed Review according Vanessa comments
	22/06/2010	Draft	Piero Corte, Ricardo Jimenez-Peris, Francisco Perez Sorrosal	The Structure of the Reference Architecture (§1.3), the Pattern Template descri (§3.5) and the Cross Cutting Pattern relationships section (§3.4) have been added.
	25/06/2010	Draft	Evelyn Pfeuffer	Internal review comments integrated

EXECUTIVE SUMMARY

This deliverable is dedicated to lay the principles and the baseline for the creation of *NEXOF-RA Specification*. It mainly fixes rules and restrictions for the formal aspects of the specification, such as the format, the structure and its development approach. As far as it concerns architectural solutions, this document does not state any restricted principle, since it is well-known that NEXOF Reference Architecture is domain and technology independent. A part from the restriction on the very general domain of SOA Infrastructures, any concern, problem and solution related to this domain generally is interesting for the project.

In the first section this document provides a description of what the NEXOF Reference Architecture is, what it is useful for and what its overall structure is. This section if fundamental to understand all the principles and baseline introduced afterwards.

The second section gives a list of principles used for the development of the specification. They have been selected as guidelines for the specification process in order to produce an open and easily evolvable specification.

The third section is dedicated to the introduction of the idea of *constructional patterns* as the basic mechanism (baseline) that is used to develop the overall specification.

Finally, the last section is devoted to the description of the *Pattern Development Process*. This process is a part of the overall specification process and covers the development of each individual pattern.

Document Information

IST Project Number	FP7 – 216446	**Acronym**	NEXOF-RA
Full title	NESSI Open Framework – Reference Architecture		
Project URL	http://www.nexof-ra.eu		
EU Project officer	Arian Zwegers		

Deliverable	**Number**	D7.2c	**Title**	Definition of an architectural framework & principles
Work package	**Number**	WP7	**Title**	NEXOF Reference Architecture Specification

Date of delivery	**Contractual**	30/06/2010	**Actual**	30/06/2010
Status	Version 0.8, dated 30/06/2010		final ☐	
Nature	Report ☐ Demonstrator ☐ Other ☑			
Abstract (for dissemination)				
Keywords	NEXOF-RA Specification, Architectural Pattern, SOA			

Internal reviewers	Evelyn Pfeuffer			
	Vanessa Stricker			
Authors (Partner)	Piero Corte, Debora Desideri, Ricardo Jimenez-Peris, Francisco Perez Sorrosal			
Responsible Author	Piero Corte		**Email**	piero.corte@eng.it
	Partner	Engineering	**Phone**	0039 06 49201416

TABLE OF CONTENTS

2 THE NEXOF REFERENCE ARCHITECTURE SPECIFICATION

This section gives a brief explanation of *NEXOF Reference Architecture* before focusing on the specification. This is necessary to clearly present what the *NEXOF-RA Specification* is. Furthermore, it will be described how NEXOF Reference Architecture is expected to be used.. The last subsection presents the structure of the *NEXOF Reference Architecture* and introduces the *NEXOF-RA Specification* by describing it with respect to the other parts too.

2.1 What NEXOF Reference Architecture is

An initial characterization of NEXOF Reference Architecture is the following: "*NEXOF Reference Architecture is a set of instruments designed to help the construction of well-architected distributed SOA software infrastructures*". This initial characterization shifts the focus towards one of the most fundamental questions: "what is a distributed SOA infrastructure?" The answer to this question is the key to fully understand what NEXOF Reference Architecture is, the type of instruments it is composed of and how it helps constructing these kind of software systems.

SOA is not a specific technology or predefined solution but rather a paradigm or architectural style that is used to improve the scalability and decentralization within distributed, heterogeneous, cross-business-domains IT environments. It is an approach to deal with an environment in which processes and systems are becoming more and more complex and IT landscapes are rapidly changing. SOA aims at closing the gap between business and IT in these environments in order to flexibly and efficiently exploit business opportunities.

Characteristics of a deployed service-oriented system are unique within each business company. Therefore there is not a one sized SOA solution that fits all situations and a SOA solution needs to be adapted for each individual context. However, a SOA solution is always characterized by the usage of technologies and platforms that specifically support the creation, execution, and evolution of services.

Like all other software system architectures, *Service-Oriented Architectures* can be captured by means of models, specifications and accompanying material. Based on these instruments a concrete implementation can be built.

The term "software system architecture" traditionally addresses the organizational structure of all elements that are part of a software system [3]. However, a software system can be differentiated into different types of elements, whose structure can be described by different architecture documents. A system provides an operational environment that allows for the deployment of applications and processes. Accordingly these systems can be divided into the infrastructure and the operative elements that can be deployed in order to focus on business objectives.

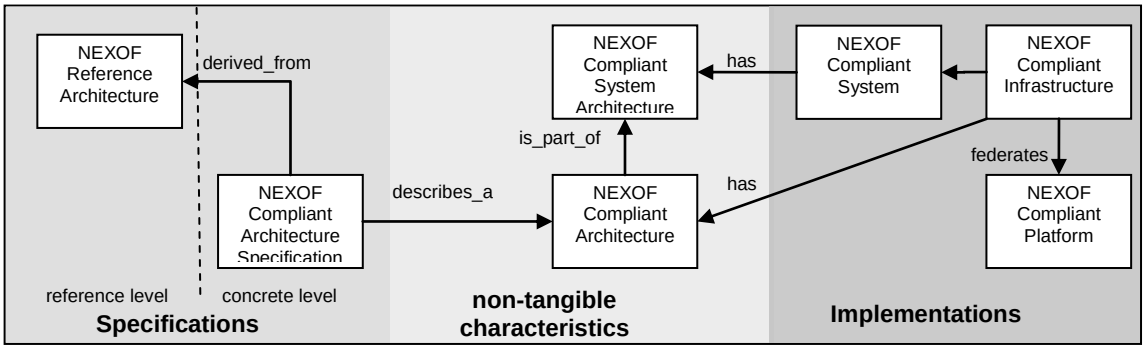


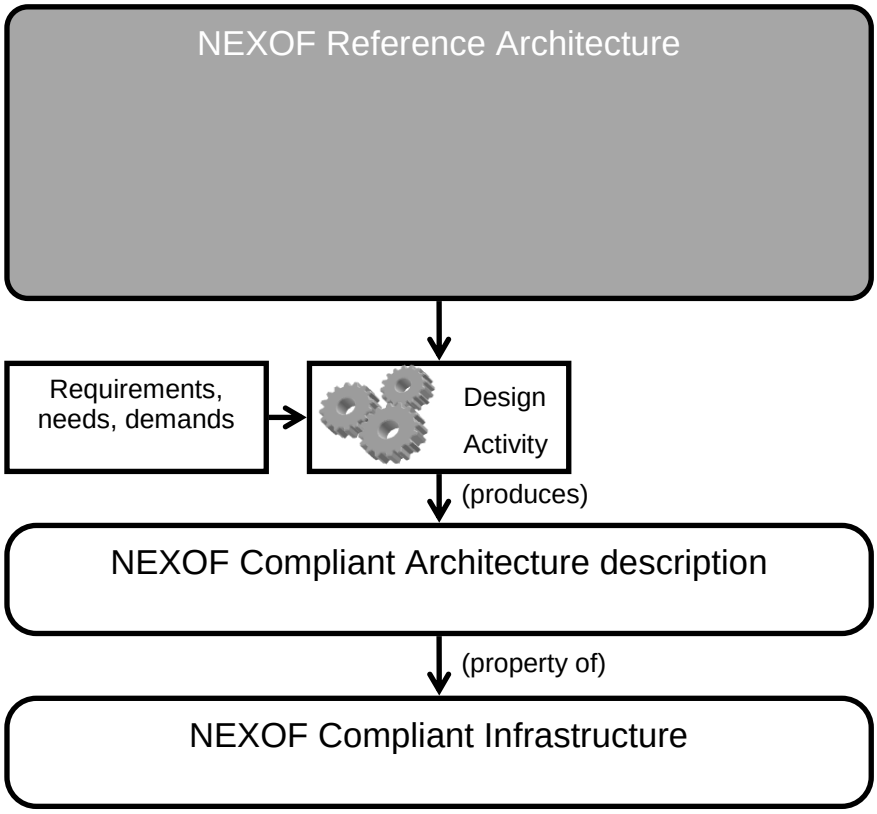
Figure 1: Architecture terminology in the NEXOF context

The SOA paradigm introduces the notion of a specific infrastructure on which services can be deployed and executed in a distributed manner. The *NEXOF-RA Project* will produce the reference architecture for such kind of infrastructures. The project only addresses the architecture of the infrastructure. Concrete applications and services are not in the focus since the reference architecture should be domain independent and open. The infrastructure architecture addresses the hardware infrastructure architecture as well as the software infrastructure architecture. Thus, this infrastructure, the *NEXOF-RA Infrastructure* can be perceived as an operating environment for services and service-oriented applications.

Concluding, the "**NEXOF Reference Architecture is a construction kit that will aid system architects in the construction of specific architectures of SOA Infrastructures**". If the architecture of a *SOA Infrastructure* follows the suggestions from NEXOF Reference Architecture, it will be called a *NEXOF Compliant Infrastructure* **(NCI)**. Based on this NCI, instances of service-based systems can be built, which from now on are called *NEXOF Compliant System* **(NCS)**.

2.2 The Purpose of the Reference Architecture

The NEXOF Reference Architecture does not provide a typical system architecture description but define a meta-level description providing a set of different alternatives from which concrete solutions can be derived as it is shown in the next figure.



NEXOF Reference Architecture is domain independent and will be accompanied with a sound methodology and tools to be properly instantiated within different contexts for a broad range of end-user communities (including Large, Medium, and Small Enterprises), and by means of different technologies.

The design of a NEXOF Compliant Infrastructure starts with the identification of the requirements addressing the needs and demands to be fulfilled in each specific context. These requirements address both the functional and non-functional properties of the system to be built and deployed. The system architect then matches these requirements to the properties of the reference elements captured in the Reference Architecture. Through this matching, the architect will select, and combine patterns to design the architecture of the system to be built and deployed.

Several different system architectures, that are **NEXOF Compliant Architectures**, can be derived from the NEXOF Reference Architecture. A NEXOF Compliant Architecture can be the architecture of several service-based software systems that are NEXOF Compliant Systems. According to this approach, NEXOF Reference Architecture will allow the building of a **NEXOF Compliant Infrastructure**.

2.3 The Structure of the Reference Architecture

The NEXOF Reference Architecture is composed of several parts as shown in the next figure, capturing the information necessary to design service-oriented software systems.

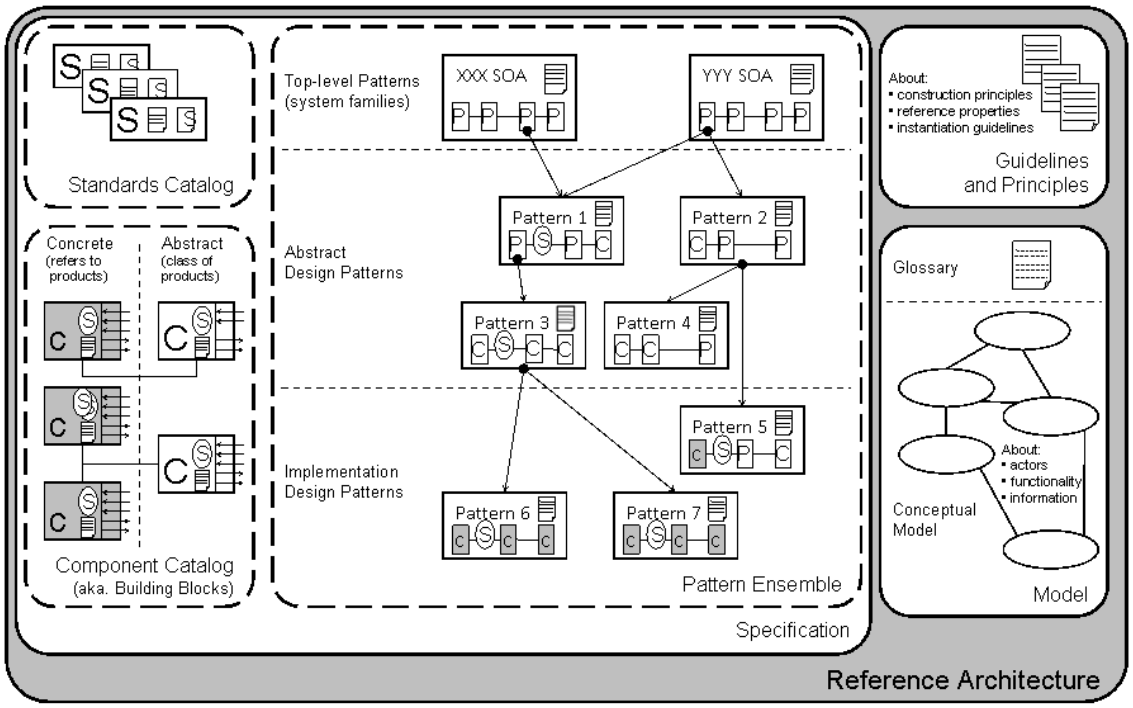


Figure 2: The NEXOF Reference Architecture Structure

The main constituents of the Reference Architecture are:

- The **Guidelines and Principles**: these captures the principles underlying the construction of the framework, the set of reference-properties associated with each of the components and patterns in the NEXOF Reference Architecture, and the guidelines used to instantiate a specific system architecture according to its requirements
- The **Reference Architecture Model**: this is the Conceptual Model which describes the essential entities that constitute service-based systems, as well as the relationships between these entities and the key elements in the context. In addition, it contains also the *NEXOF-RA Glossary [8]* which defines the terms used across the whole *NEXOF-RA Project*. This glossary has been built and is maintained under the auspices of the *NEXOF-RA Architecture Board* in which representatives of all NESSI Strategic Projects are included
- The **Reference Architecture Specification**: This contains three collections:
  - The *NEXOF-RA Standards Catalogue*[1]: the standards referred to in the Reference Architecture are described in this catalogue. Each standard is linked to the relevant elements of the Guidelines and Principles as well as to the concepts it addresses

[1] NEXOF-RA Standards Catalogue, http://www.nexof-ra.eu/?q=node/529

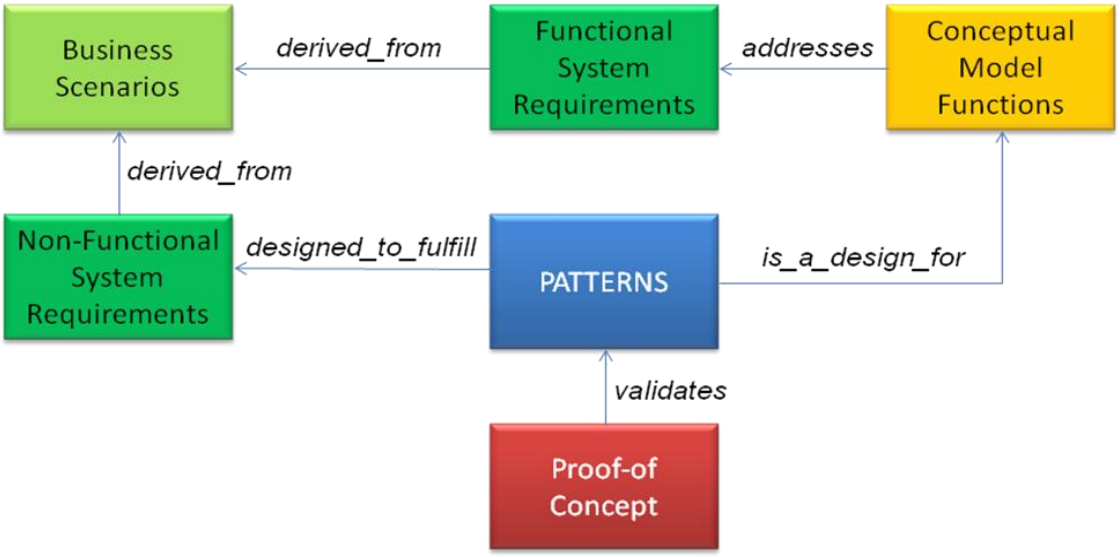
- The *NEXOF-RA Component Catalogue*: the level of granularity considered in the Reference Architecture is that of components, which roughly correspond to coherent sets of functionality delivered as software products or software components which can be configured separately. This catalogue groups both abstract descriptions of components (e.g. an UDDI registry) and product or software-based components (e.g. the jUDDI library). Each description refers to the standards it implements, the concepts it addresses, as well as its behavioural characteristics
- The *Pattern Ensemble*[2]: the actionable part of the Reference Architecture is represented by the patterns, which define various ways of realizing some functionality by associating components and other patterns in a defined manner.

The *NEXOF-RA Specification* defines a system of patterns (cf. Alexander, C. [3], and Buschmann et al. [5],[1]) that includes three types of patterns: top-level patterns which describe the characteristics of SOA platform families, abstract design patterns which refer to abstract components and patterns, and implementation design patterns which refer to at least one concrete component. Relationships between patterns (e.g. “extends” or “isPartOf”) are explicitly described. Each pattern description also refers to the standards it implements, the concerns it addresses, as well as its behavioural characteristics.

The conceptual view plays a decisive role within the creation of the specifications itself as well as in the design of NEXOF Compliant Architectures since it allows the communication about the relevant components on a higher abstraction level. It is important to understand on a conceptual level what has to be specified as well as the general relationships between those elements. This conceptual view will be captured by the NEXOF Reference Architecture Model comprising also the glossary which represents the terminological definitions.

The following picture shows how different results of the NEXOF-RA project, such as the business requirements, the conceptual model, the architectural patterns and the proof-of-concepts are inter-related:

[2] Pattern Specifications, http://www.nexof-ra.eu/



In particular, patterns are relates indirectly to Business Requirements, since they **are designed for** fulfilling the functionality formalized by the Conceptual Model that **addresses** and covers all the functionalities of the service-based software system platform **derived** from the business scenarios.

Moreover, patterns are also **designed to fulfill** the Non-Functional System Requirements of such a service platform **derived** from the business scenario.

Finally, patterns are inversely-relates to the POCs, since POCs **validate** the truth of the Quality Attributes Trade-off Evaluation Statements claimed by the patterns.

3 SPECIFICATION PRINCIPLES

The *NEXOF-RA Specification* is developed using a semi-formal approach, i.e. its parts (software architecture patterns) are provided by mixing natural language descriptions and UML specification techniques. While this approach lacks some of the rigor of a formal specification method, it offers the advantages of being more intuitive and pragmatic for most system architects and practitioners.

The following items summarize the principles followed to build the *NEXOF-RA Specification*:

- **Modularity** — This principle of strong cohesion and loose coupling is applied to group parts of the specification into different categories (i.e. SOA Domains). Describing architectural problems in terms of patterns adheres to the concept of modularity especially since components included in these patterns will be separated by well-defined interfaces.
- **Layering** — Layering is applied to separate top-level architectures descriptions (i.e. architectural patterns) from the more specific and detailed parts (i.e. design and implementation architectures descriptions). Furthermore, the concept of layering can be found in different patterns as architectural choices to address specific problems.
- **Partitioning** — Partitioning is used to organize conceptual areas (i.e. SOA concerns) within the same layer. All concerns that have been identified in the model will be addressed by patterns of all layers.
- **Extensibility** — The NEXOF Reference Architecture is developed in a way that it can be easily extended and enhanced. In particular, the specification will be constituted of related parts. These relationships will make possible to have parts that refine others or provide an alternative solution to others. This way, it will be always possible to extend the specification by adding new parts without affecting the already included ones.
- **Reuse** — It is possible to add new parts to the specification by reusing parts that are already included into it.

4 SPECIFICATION BASELINE

One of the fundamental principles of software engineering that is largely used when designing a software system is known as the “separation of concerns”. In short, this principle states that a larger problem is more effectively solved when decomposed into a set of smaller and independent problems or concerns. This gives software engineers the option of partitioning solution logic into capabilities, each designed to solve an individual concern. Related capabilities can be grouped into units of solution logic. The main benefit to solving problems this way is that a number of the solution logic units can be designed to solve immediate concerns while still remaining agnostic to the greater problem. This provides the constant opportunity for us to reuse the capabilities within those units to solve other problems as well.

The approach adopted in *NEXOF-RA Project* is also based on the separation of concerns principle. *NEXOF-RA Project* objective is to address the problem of specifying SOA-based software system architectures by partitioning the overall solution into several pieces of designing solution: **patterns**. In particular, we recognize the need for a development methodology to develop large-scale complex systems and, at the same time, learn from the experiences of other systems designers in solving recurring design problems.

Usually **a pattern**, as it stands, **describes the architecture of a system**, providing details about which parts the system is composed of, the role of these parts, how these parts are integrated and cooperate, forces and consequences, and guidelines for implementation [3]. Furthermore the solution provided by a pattern is given in terms of architectural choices and statements that claim how these architectural choices affect the quality attributes of a system that is compliant to such a pattern. *NEXOF-RA Project* approach is also focused on **how to compose these patterns together** to develop complete systems [13][14]. A complete system cannot, nor will ever, be built from a single pattern. It is the integration and composition of patterns that makes a whole system.

A pattern can be considered as a design element with interfaces (i.e. the interfaces of the system it describes), in this sense it can be used to represent the system it describes. For this reason, our approach will adopt patterns as first class design elements. I.e. they can be used in the designing of a system as any other design element: class, module, component etc. These kinds of patterns are called **constructional-patterns**. This way, patterns can be designed as a composition of other patterns. Accordingly, “a pattern describes a system that is designed through the composition of systems described by other patterns”.

Specifying a pattern as a design element leverages the value of a pattern to a higher design level that hides later design details and preserves consistency with lower levels.

Patterns constitute the mechanism that WP7 has chosen to develop the *NEXOF-RA Specification*. In particular the **constructional-pattern-based approach** adopted in the NEXOF Reference Architecture constitutes the proper

answer to fulfil the architectural specification principles described in the previous section.

4.1 What is a NEXOF-RA Pattern?

A *NEXOF-RA Pattern* is the description of the architecture/design /implementation of a system. In particular it provides a description of:

- the solution, i.e. the components the system is composed of, the role and the functionalities of these components, how these components are integrated and cooperates
- the problem that the solution is designed for and the context where the solution is applicable (forces)
- the consequences of its application, i.e. how the application of the pattern affects the property of the system or more general the trade-off evaluation of its quality attributes.

In NEXOF Reference Architecture we take apart patterns in three levels of abstraction:

- **Top-level** patterns: they are *architectural* patterns. They express a fundamental structural organization schema for a complete service based platform. It provides a set of predefined subsystems or components, specifies their responsibilities, and includes rules and guidelines for organizing the relationships between them [1]. Architectural patterns are a means of documenting architecture for complex and heterogeneous systems, thus helping to manage the application complexity.
- **Abstract** patterns: they are *design* patterns. They provide a schema for refining the subsystems or components of a software system or the relationship between them [1]. It describes a commonly recurring structure of communicating components that solves a general design problem within a particular context [5].
- **Implementation** patterns: they are *idiom*s. They are low-level patterns specific to a technology: standard, programming language, product. An idiom describes how to implement particular aspects of components or the relationships between them using the features of a given technology.

With respect to this general classification the *NEXOF-RA Specification* will result in a consistent collection of constructional architectural/design/idiom patterns.

The top-level patterns play a particular role in the application of the design methodology as they define different classes (or families) of SOA platforms that will be implemented. The principle of independence from application domains implies that the NEXOF Reference Architecture must allow the instantiation of NEXOF Compliant Architectures for many different application domains, such as enterprise systems, manufacturing systems, real-time systems, sensor networks, and automotive communications. In essence, the properties of Service-Based Systems used for each of these applications is different enough

from the others that it constitutes an own system type. However, once the top level pattern has been identified, its association with abstract and implementation design patterns follows a strict structure of dependencies and refinements.

4.2 Type of patterns: functional and cross-cutting

Patterns within the *NEXOF-RA Specification* can be classified also according to the following categories:

- Functional / Non-Functional Patterns;
- Cross-Cutting / Non-Cross-Cutting Patterns;

A *Functional Pattern* provides an architectural solution to a software system that is mainly characterized (constrained) by its functionality requirements, i.e. the functionalities it must provide. As a consequence, the *NEXOF-RA Specification* is expected to contain more Functional Patterns that provide different architectural solutions to the same set of functional requirements (problem).

A *Cross-Cutting Pattern* is a pattern that is applicable to other patterns (see below the *isApplicableTo* relationship section). Basically, it provides a means (the guidelines) to transform and enhance the architectural solution provided by another pattern. By definition, all those patterns that are not applicable to others are Non-Cross-Cutting Patterns. A Cross-Cutting Pattern can be either a Functional Pattern or a Non-Functional Pattern.

A Non-Functional Pattern is a Cross-Cutting Pattern that is specifically design to transform the architectural solution of other patterns (those it can be applied to) to improve some of its quality attributes.

4.3 Collection of related patterns

Most pattern definitions do not - at least not explicitly - emphasize the many relationships that exist among patterns. Understanding and using individual patterns as isolated islands without considering their connections to other patterns may help to resolve localized problems, but rarely more. Such isolation means that large-scale design problems persist, small-scale problems recur without resolution, and the architectural vision of the overall system remains unclear to developers. Acting both globally and effectively with individual patterns becomes hard in the absence of such an all-encompassing vision. For this reason, the *NEXOF-RA Project* approach also aims at producing a pattern map to show relationships between all the patterns produced.

Therefore each pattern must provide the relationships that the proposed pattern has got with other patterns, so that they can be used to scale beyond point solutions, to address larger and more sophisticated problem spaces.

Here after are reported all the kinds of identified relationships.

4.3.1 Extends Relationship

The "extends" relationship can be used to define a pattern as a refinement of another one.

Formally, a pattern R *extends* a pattern P (R extends P) when:

- R meets the functional requirements of P,
- R assumptions are a subset of P assumptions and
- R makes different architectural choices from P.

It follows that R can be used anywhere P can be used (not vice versa).

A graphical representation of the “extends” relationship can be seen in the picture sotto.

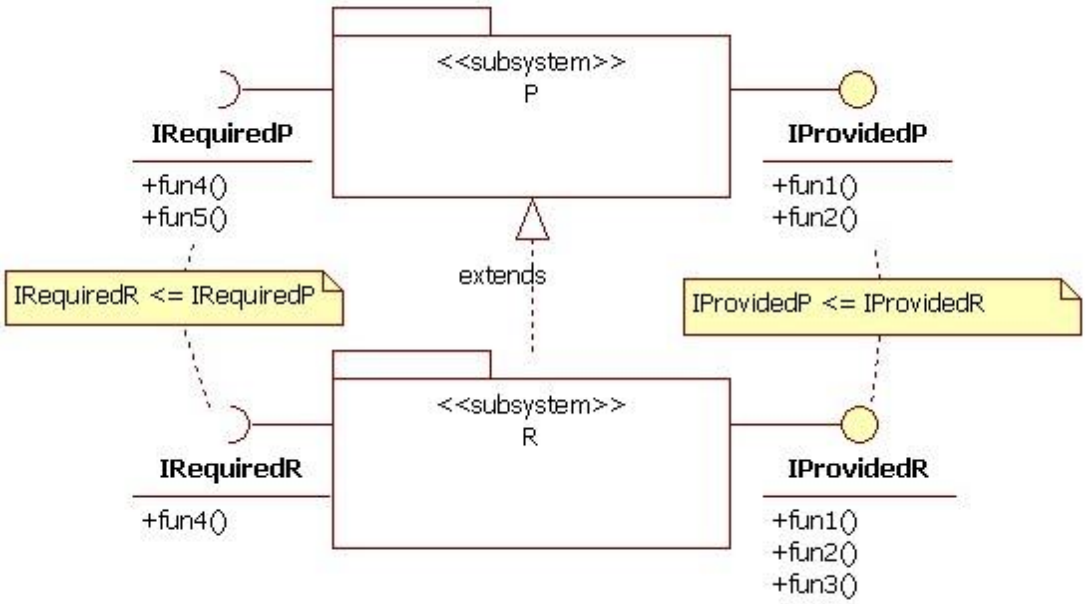


Figure 3: Extends relationship example

4.3.2 IsPartOf Relationship

The “isPartOf” relationship can be used when a pattern is composed of other patterns.

Formally, a pattern R *isPartOf* a pattern P with respect to a set of components Ci of P (*R isPartOf[C1,C2,…] P*) when

- R provides an architectural solution for such set of components.

This implies that the solution provided by the pattern R has to satisfy all the requirements that such set of components must meet in P.

Hereafter two examples of the application of “isPartOf” relationship are shown.

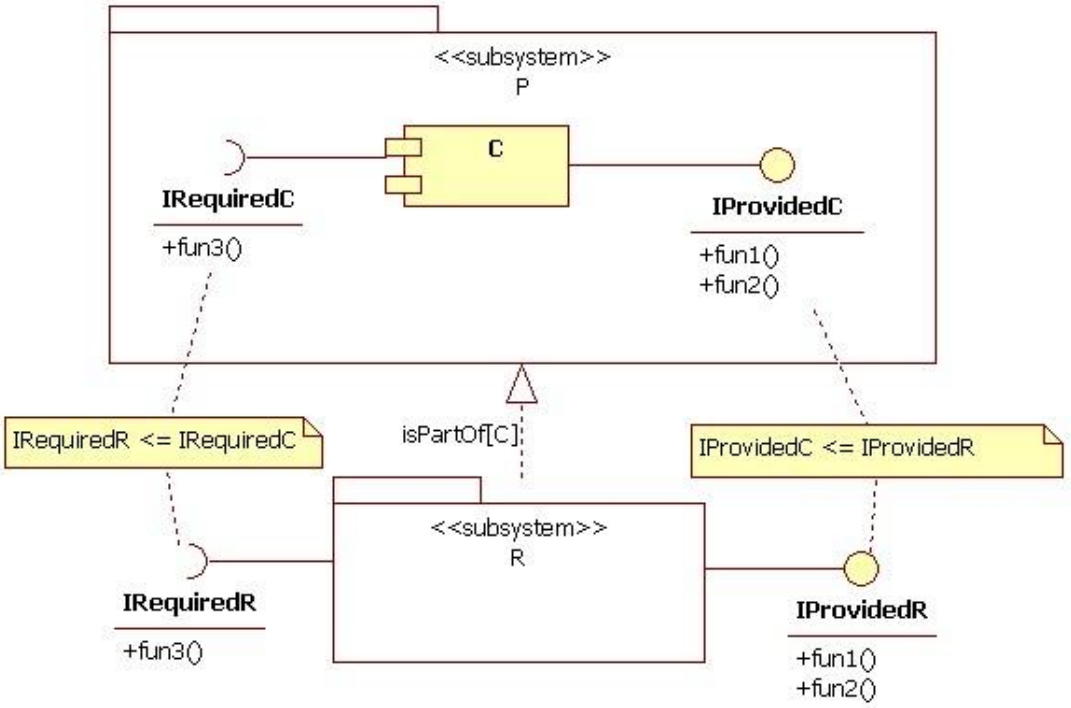


Figure 4: isPartOf relationship example 1: *R isPartOf[C] P*

Figure 4 shows a pattern R that *isPartOf* a pattern P with respect to a component C part of the solution described by P (*R isPartOf[C] P*).

While the component C in the pattern P is a black box component, the pattern R provides an architectural solution for the component C. In general several patterns describing the architectural solution of a same component C in a different way can exist.

In this example the required and provided interfaces of the pattern R are the same of the required and provided interfaces of the component C. More in general the functionality provided by R must “cover” the functionality provided by C, this means that the interfaces can be different but it is explicitly stated which functionalities of R corresponds to (by means of specialization or composition relationships) each functionality of C. As far as it concerns the functionality required, in general the functionality required by R can be less then those required by C, this means that R requires less constraints then C on the context where it can be applicable.

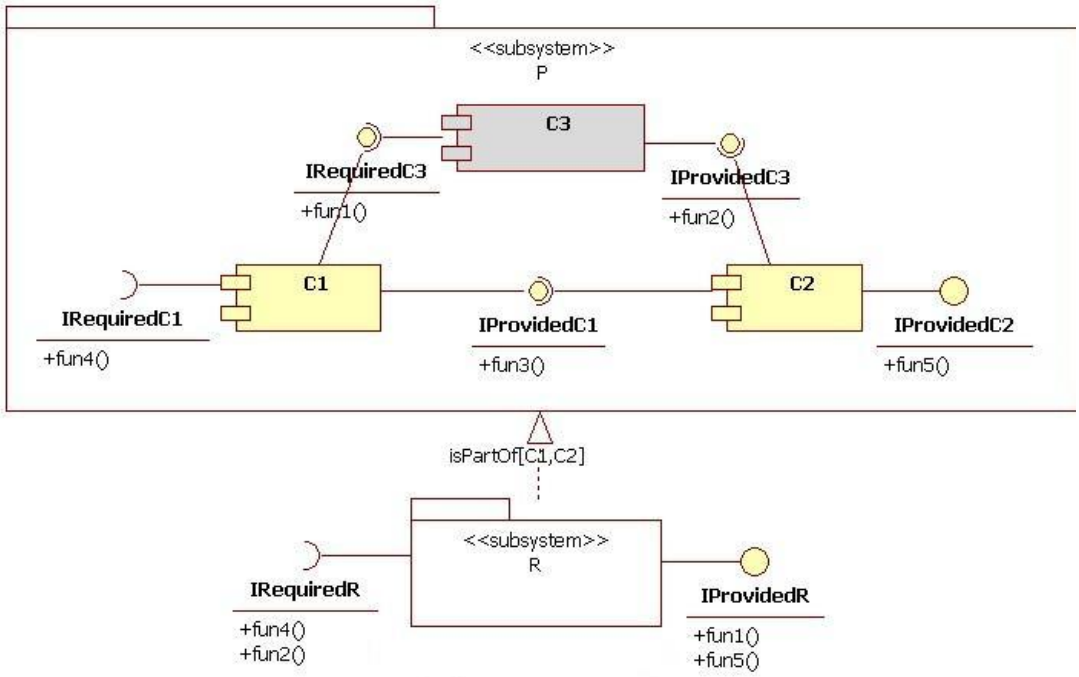


Figure 5: isPartOf relationship example 2: *R isPartOf[C1,C2] P*

Figure 5 shows the "isPartOf" relationships with respect to 2 different components, in particular a pattern R that *isPartOf* a pattern P with respect to the components C1 and C2 that are part of the solution described by P (*R isPartOf[C1,C2] P*).

In this case, the required and provided interfaces of the R must correspond to the functionalities that C1 and C2 together require and provide to the rest of the components in P respectively.

4.3.3 ComplementsWith Relationship

The "complementsWith" relationship is used to state when two patterns describe two architectural solutions that are well-designed to be jointly applied into the design of an overall system.

Formally a pattern R *complementsWith* a pattern P in the context of a pattern Q (*R complementsWith[Q] P*) when

- a pattern Q exists and
- P *isPartOf*[C1] Q,
- R *isPartOf*[C2] Q and
- P and R are strongly recommended to be used concurrently to realize Q.

In other words, R and P provide two specific and complementary parts of the solution of Q.

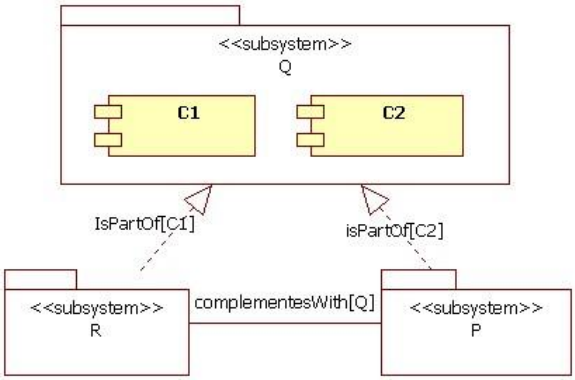


Figure 6: complementsWith relationship example: *R complementsWith[Q] P*

4.3.4 CompetesWith Relationship

The "competesWith" relationship shows that instances of two patterns should not be used together, that is, they provide architectural solutions that should not be concurrently used into the design of an overall system..

Formally a pattern R *competesWith* a pattern P in the context of a pattern Q (*R competesWith[Q] P*) when:

- a pattern Q exists and
- P *isPartOf*[C1] Q,
- R *isPartOf*[C2] Q but
- P and R cannot be used concurrently to realize Q.

In other words, R and P provide two mutually exclusive part refinements of the solution of Q.

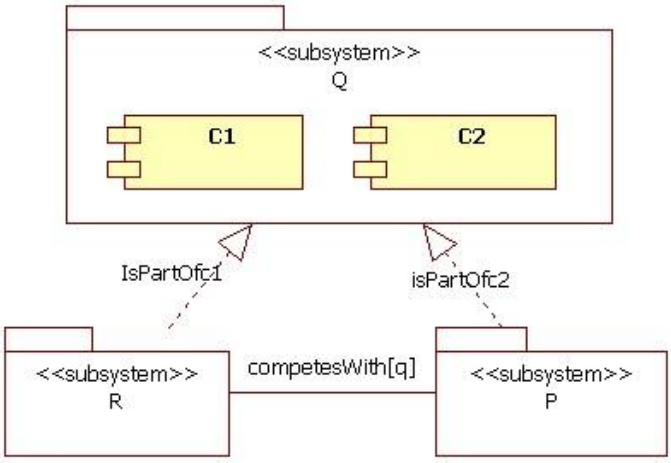


Figure 7: competesWith relationship example: *R* competesWith *[Q] P*

4.3.5 IsApplicableTo Relationship

The “isApplicableTo” relationship is used when a pattern can be “applicable to” the design solution provided by another pattern.

Formally, a pattern R is applicable to a pattern P (R *isApplicableTo* P) when:

- The architectural solution provided by P fulfils the assumptions stated by R.

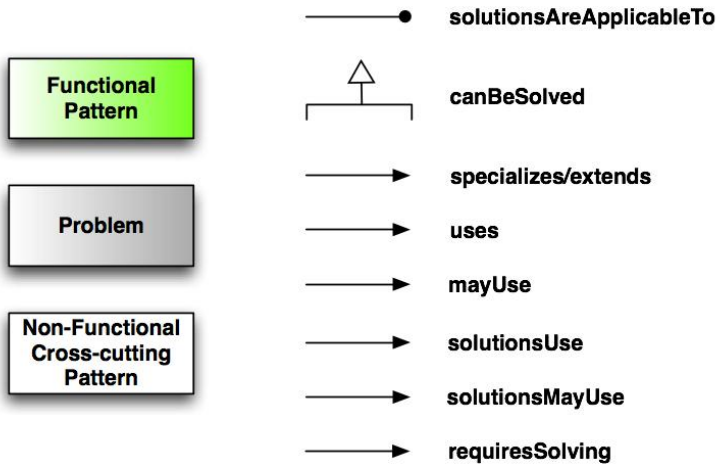
Within the *NEXOF-RA Specification* this relationship occurs when it is explicitly stated between two patterns or it can be inferred when the matching between the design solution of one pattern and the assumption of another occurs.

This relationship is very powerful. It allows the development of cross-cutting patterns that can be applied recursively to other patterns, keeping orthogonal and minimal the *NEXOF-RA Specification*, but still capable to produce several different architectural solutions.

4.4 Collection of related cross-cutting patterns

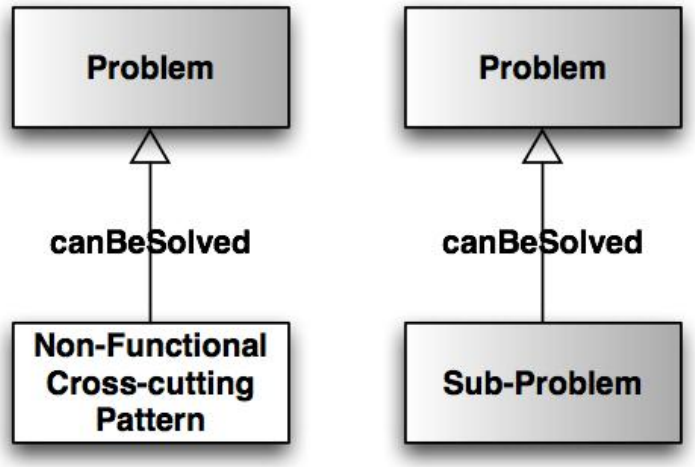
Cross-cutting patterns due to their cross-cutting nature do not have the same relationships as non cross-cutting ones. The reason is that they might apply to many patterns, i.e., all that fulfil their assumptions. In order to express their special relationships we need a new entity that we will term *problem*. In this way, cross-cutting patterns that aim to solve the same problem might be related whilst with the previous set of relationships they would appear as unrelated. This enables to represent taxonomies of cross-cutting patterns by relating them through the problems that they solve. The second element that is missing from the former relationships with respect cross-cutting patterns consists of the relationships between cross-cutting and problems and between cross-cutting patterns themselves. The following figure shows the new kind of boxes and relationships that will be introduced for better interrelating cross-cutting patterns:

Legend for categorizing a set of non-functional cross-cutting patterns



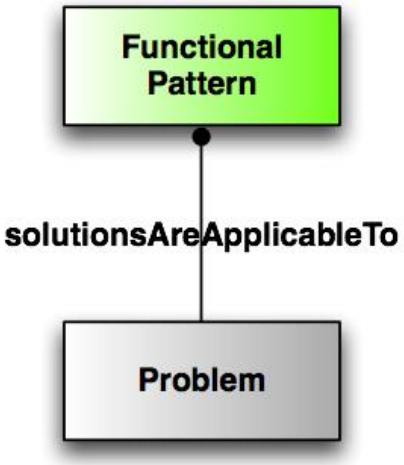
B*oxes* represent patterns and problems to be solved and *lines* represent new kinds of relationships. Because of cross-cutting patterns are organized by the high level problems that they solve, there is also the need to highlight which functional patterns provide the context for the high level problems. The functional pattern(s) that provide this context when describing a problem domain are highlighted in green. The non-functional cross-cutting patterns are depicted as *white boxes*. G*ray boxes* represent the problems that the patterns aim to solve. In the following we describe the new relationships that have been introduced to relate cross cutting patterns with problems and among themselves. As it has been stated before, these relationships joint with the problem boxes, enable to represent with clear semantics the taxonomies of cross-cutting patterns by means of pattern diagrams that system architects can perfectly understand.

4.4.1 canBeSolved Relationship



This relationship is represented as a white-headed arrow. It is used to categorize alternative solutions for a problem. If the relationship connects a cross-cutting pattern to a problem (the white-headed arrow points to the problem), it means that the pattern can be applied to solve that particular problem. If the arrow connects a problem to a sub-problem (the white-headed arrow points to the more general problem), it means that the solutions to the sub-problem (i.e. the pattern/s) are also valid to solve the more general problem (transitive relationship).

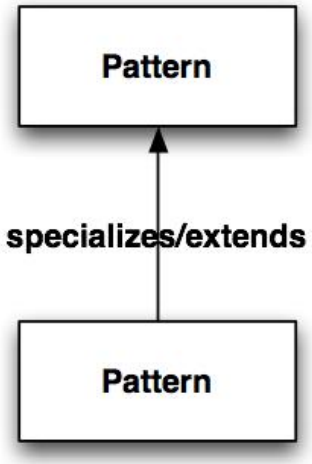
4.4.2 solutionsAreApplicableTo Relationship



This relationship is represented as a line ended in a black dot. It connects a functional pattern with a problem that *canBeSolved* with one or several non-functional cross-cutting patterns (See *canBeSolved* relationship). It means that the solutions to the problem provided by the non-functional patterns are applicable to the functional pattern.

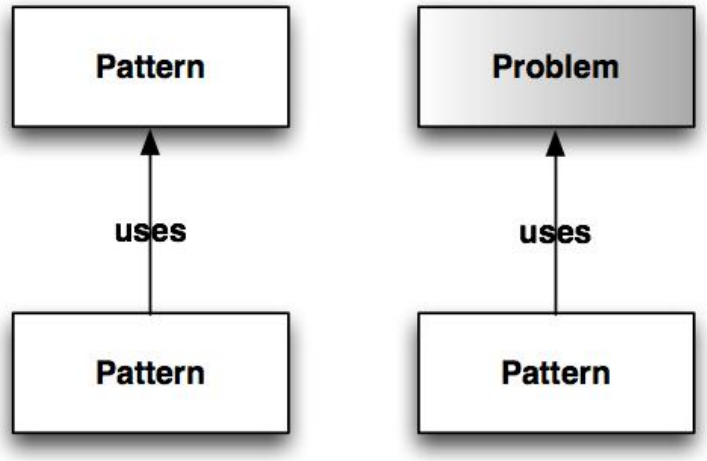
The rest of the relationships are represented with arrows, and express the following semantic relationships:

4.4.3 specializes/extends Relationship



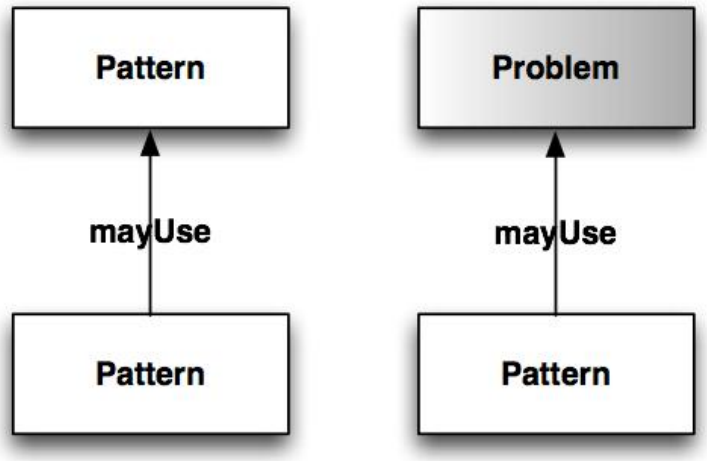
This relationship connects a pattern to another pattern. The patterns can be either functional or not and cross-cutting or not. This relationship is equivalent to the one described in Section 4.3.1 and it has been included here for the sake of completeness of the relationships related to cross-cutting patterns. It means that the pattern, that is the origin of the arrow, specializes/extends the features provided by the pattern the arrow is pointing to.

4.4.4 uses Relationship



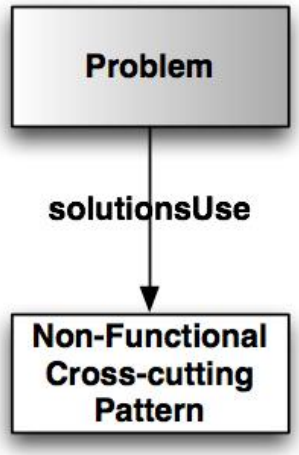
This relationship can connect a pattern to another pattern or a pattern to a problem. The patterns can be either functional or not and cross-cutting or not. In the first case, it means that the pattern that is the origin of the arrow must use of the pattern the arrow is pointing. In the second case, the relationship means that the pattern that is the origin of the arrow must use any of the solutions provided by the problem the arrow is pointing (i.e. the pattern/s that can solve the problem. See *canBeSolved* relationship).

4.4.5 mayUse Relationship



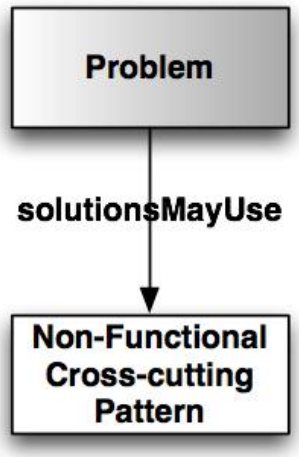
This relationship can connect a pattern to another pattern or a pattern to a problem. The patterns can be either functional or not and cross-cutting or not. The meaning in both cases is similar to the *uses* relationship, but the use of the pointed pattern or the pattern/s that solve the pointed problem is optional (See *canBeSolved* relationship).

4.4.6 solutionsUse Relationship



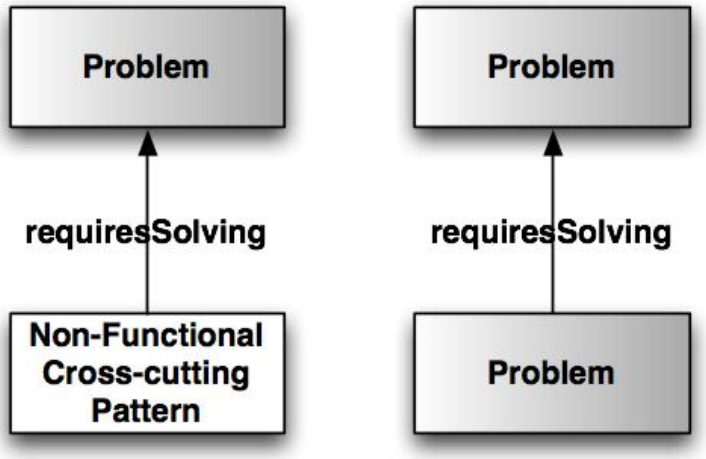
This relationship connects a problem (origin of the arrow) to a non-functional cross-cutting pattern (target of the arrow). It means that the pattern the arrow is pointing must be used by any of the solutions specified to the problem (i.e. the pattern/s that can solve the problem. See *canBeSolved* relationship).

4.4.7 solutionsMayUse Relationship



This relationship connects a problem (origin of the arrow) to a non-functional cross-cutting pattern (target of the arrow). It means that the pattern the arrow is pointing can be used by any of the solutions specified to the problem (i.e. the pattern/s that can solve the problem. See *canBeSolved* relationship).

4.4.8 requiresSolving Relationship



This relationship can connect a problem to another problem or a non-functional cross-cutting pattern to a problem. In the first case, it means that the solutions (i.e. pattern/s) for the problem that is the origin of the arrow need to solve the problem the arrow is pointing using the most adequate solution provided (See *canBeSolved* relationship). In the first case, it means that the pattern that is the origin of the arrow need to solve the problem the arrow is pointing using the most adequate solution provided (See also *canBeSolved* relationship).

4.4.9 Example

In this section we provide some concrete examples of the new boxes and relationships and how they can be used to present cross-cutting architectural alternatives to a problem domain.

The following figure (Figure 8) presents a fraction of the diagram that categorizes the non-functional cross-cutting patterns related to high availability and scalability in the context of multi-tier architectures. More specifically, it depicts the problems and patterns in the Multi-Tier Replication Domain and how they are interrelated (see the instantiation guidelines related to High Availability and Scalability described in D14.1[3]).

[3] D4.1 Multi-Tier Replication Domain in the HA and Scalability guidelines, http://www.nexof-ra.eu/

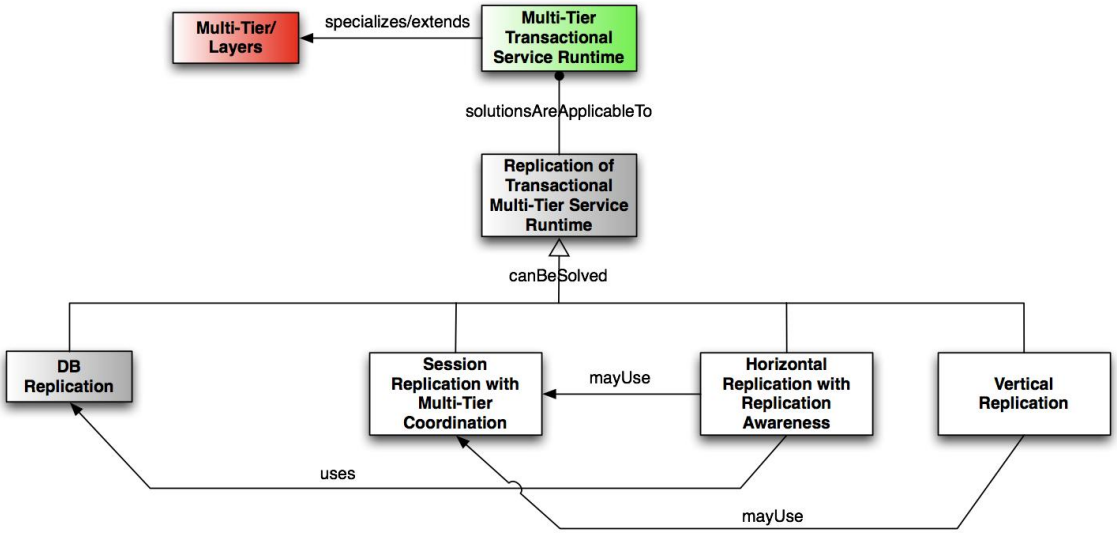


Figure 8 Multi-Tier Replication Domain in the HA and Scalability guidelines[4]

The *Multi-Tier Transactional Service Runtime* pattern (depicted in green) is the functional pattern –offering a concrete architectural approach- where the non-functional crosscutting patterns in this domain (in white) can be applied. The root pattern *specializes/extends* an architectural approach described as a pattern in the existing literature, in this case the *Multi-Tier/Layers* pattern (in red) [4][5]. The gray-box under the green box represents the main *problem* that the non-functional cross-cutting patterns presented aim to solve and can be seen as a domain for categorizing different solutions; in this case, the replication of transactional multi-tier runtimes. The *solutionsAreApplicableTo* relationship –expressed through the line ended with a black dot- that links the problem with the functional pattern, describes that the solutions to this problem are applicable to the *Multi-Tier Transactional Service Runtime* pattern in order to extend it with the additional non-functional features provided by the cross-cutting patterns. And, how can this problem be solved? The answer is in the different patterns linked to the problem through the white headed arrows (*canBeSolved* relationships). In this case, three patterns can be applied (shown in white). Moreover, there are other semantic relationships among these patterns (shown as black arrows) that express for example that when the *Vertical Replication* pattern is used, the *Session Replication with Multi-Tier Coordination* may also be used if necessary (See *mayUse* relationship). Finally, the DB Replication problem expressed as a problem box is used to categorize a new set of solutions related to database replication (See *canBeSolved* relationship). The following figure (Figure 9) presents the problems and patterns in the Database Replication domain and how they are interrelated, which can be interpreted as it has been explained above.

[4] D4.1 Multi-Tier Replication Domain in the HA and Scalability guidelines, http://www.nexof-ra.eu/

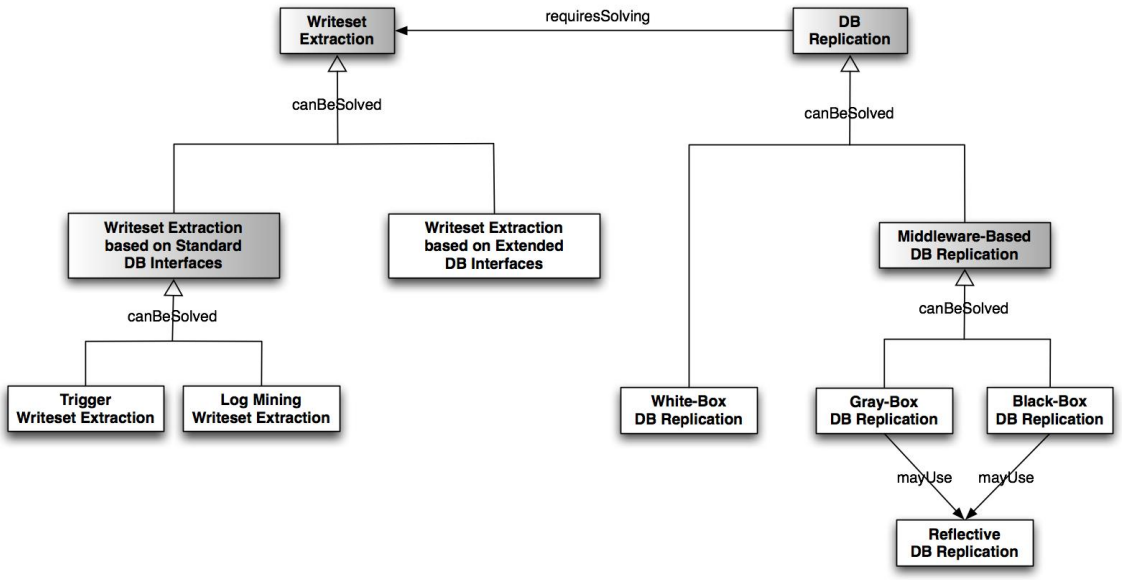


Figure 9 DB Replication Domain in the HA and Scalability guidelines (D14.1)

4.5 Pattern representation and description

The following table shows the main parts that describe a *NEXOF-RA Pattern*:

Pattern ID-Card	
Problem Description	→ Design Problem Statement
Functional Requirements	
Assumptions	
Solution	→ Architectural Solution
Non-Functional Qualities	→ Quality Attributes Trade-off Evaluation
Relationships to other Patterns	
Relationships to Components Catalogue	
Relationships to Standards Catalogue	
Application Examples	
References	

Before describing each section, it is important to highlight the essential content that is expected to be included in each pattern description. They are:

- The **Design Problem Statement** – It includes the
  - Functional Requirements – the functional description of the software system (as a black-box) the pattern is providing a solution

  - Context Assumptions – the set of constraints on usage context of the system, i.e. where the system is expected to operate
  - Non-Functional Requirements – the set of desired-effects on the quality attributes of the system, i.e. the pattern architect design goals
- The Architectural Solution – The set of architectural choices that constitute the architectural solution for the software system characterized by the problem statement
- The Quality **Attributes Trade-off Evaluation** – the set of statements that evaluate the trade-offs related to how the architectural choices affects the quality attributes

Here after a detailed description of each part is reported.

4.5.1 Pattern id-card

This part is intended to provide some description elements that are useful to identify and classify the proposed pattern.

In particular, the identified description elements are the following:

- Name: it should convey the essence of the pattern briefly.
- Abstract: it contains a short description of the pattern.
- Credits: it is the list of persons and projects that have provided the patter.
- Level: it provides the level of abstraction of the pattern. The possible values are “Top-Level” (Architectural Patterns), “Abstract” (Design Patterns), “Implementation” (Idioms) (see section 4.1).
- Type: it provides the type of the pattern. The possible values are “Functional”, “Functional-Cross-Cutting”, “Non-Functional-Cross-Cutting” (see section 4.2).
- Key Concerns: it expresses which functional concerns (see [7] for more details) the pattern covers. The possible values are Services, Messaging, Discovery, Composition, Analysis, Presentation, Management, Security, Resources. If the pattern is “Non-Functional”, the key concern is empty.
- Contact: it provides the contact person for the pattern.

4.5.2 Problem description

This section is intended to describe the problem statement for which the pattern is expected to provide a solution. It can be described by reducing the problem to sub-problems and by highlighting the major difficulties and criticisms there are in conceiving the solution. A scenario that illustrates the design problem and how the pattern solve the problem can also be presented to help understand the more abstract description of the other sections. This section can also provide a description of the situations (context) in which the problem occurs, extending the plain problem-solution dichotomy.

4.5.3 Functional requirements

This section is intended to specify the functionalities provided by the (sub-)system designed by the proposed pattern. Such functionalities must be described by UML use case diagrams together with a textual description.

The functionalities must be specified by referring those captured by the *NEXOF-RA Model* [7] .

The functionalities described in this section correspond to the provided functionalities of the overall (sub-)system and must be reflected in the "Solution" section.

Moreover if this pattern is related to other patterns (i.e. it states relationships with other patterns in its "Relationships to other patterns" section) the content of this section must be worked out with respect to the constraints imposed by the kind of relationships the pattern states.

Names play a fundamental role in describing functionalities, that is they are canonical references to the functionalities of the SOA Infrastructure functionalities as captured by the *NEXOF-RA Model* [7].

Any functionality renaming, decomposition and/or specialization must be clearly captured and described in this section by introducing the new functionalities and properly relating them to those coming from the model or the related patterns.

If the proposed pattern is a "Not-Functional" one, this part can be empty.

4.5.4 Non-functional qualities (quality attributes)

This section is intended to provide an evaluation of the quality attributes affected by the (sub-)system covered by the proposed pattern. For each quality attribute it must be indicated if the pattern affects positively (+) or negatively (-) the attribute. The fully described list of admitted quality attributes can be found in the "Quality Model for NEXOF-RA Pattern Designing report" [9]. Such a document is open and new attributes can be added if needed.

E.g.:

Availability	+
Adaptation to new operating environments (Portability)	-
Performance (efficiency)	-
Scalability	+
Security	+

Table 1: Non-functional qualities

Moreover this section must provide a detailed description about how the **design choices** stated by the pattern **affect** the **quality attributes** listed above. Design choices are performed to meet specific design goals such as: Decomposition, Replication, Compression, Abstraction, Resource Sharing, Self-Monitoring,

etc…These design goals, also referred as design operators [10], are very important because they do represent the principles that help the architect decompose the system into components and connectors that achieve desired quality attributes. Applying these principles transforms the architectural design of a system into one that is functionally equivalent but which exhibits different quality attributes. They apply to the component view of the architecture and effectively affect a subset of quality attributes, some positively and some negatively. Hereafter some examples of basic correspondence of design goals and the quality attributes they affect are provided [12]:

- **Decomposition** (Modularity) improves Scalability, Modifiability, Integrability, Portability, and Reusability, and also influences Performance and Buildability
- **Replication** improves Reliability, influences Performance and negatively affects Modifiability, Portability, Buildability and Reusability
- **Compression** improves Performance, influences Buildability and negatively affects Scalability and Modifiability
- **Abstraction** (Contracts) improves Modifiability and Portability, and negatively affects Performance
- **Resource Sharing** improves Modifiability, Portability, influence Performance and Reusability

Other correspondences can concern design goals that deal with connectors[11] (the way components interact). Some examples are:

- Client-Server: improves Performances and Resource efficiency, and negatively affects Modifiability
- Peer-to-Peer : improves Performances and negatively affects Modifiability
- Publish-Subscribe: improves Modifiability, and negatively affects Performances

Note that the examples just provided are intended to be neither exhaustive nor precise. They only aim to give a rough idea of what kind of architectural decision (design goals) should be explicated in a pattern and the effects on the quality attributes.

4.5.5 Assumptions

This section is intended to precisely describe the context where the proposed pattern can be applicable. It must provide the functionalities and the non-functional properties the pattern assumes to be respectively offered and met by the other parts of the system before it makes sense to apply the pattern. The functionalities must be selected from the RA Model [7], the non-functional properties must be expressed according to the "Quality Model for NEXOF-RA Pattern Designing report" [9]. The functionalities described in this section correspond to the required functionalities of the overall (sub-)system and must be reflected in the "Solution" section.

In most cases this is sufficient.

In case of a Cross-cutting pattern (both “Functional-Cross-cutting” and “Non-Functional-Cross-Cutting”), this section must also provide the basic pattern which the proposed pattern is applicable to.

For instance, suppose you want to provide a pattern P that aims to show how to improve scalability on a multi-tiered pattern. This section should introduce the architecture of the basic pattern (next picture) the pattern P can be applied to.

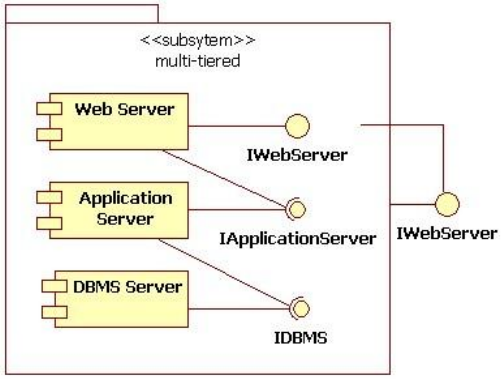


Figure 10: Multi-tiered pattern example

Then the Solution section must show how to modify it to improve scalability.

The following picture (that is only an example that should be contained in the Solution section) shows a pattern P that describes how to modify the multi-tiered pattern to improve its scalability.

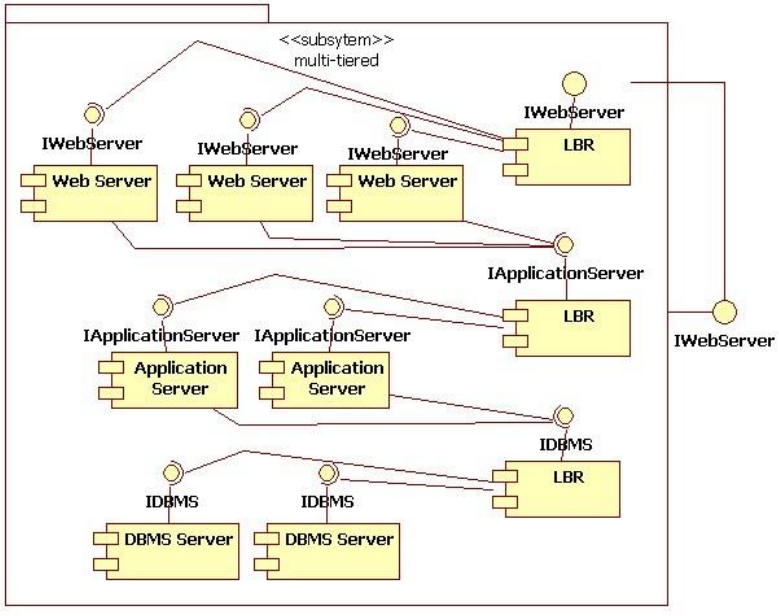


Figure 11: Scalability pattern applied to a multi-tired pattern

4.5.6 Solution

This part is intended to contain the design solution proposed by the pattern to solve the problem and fulfil the requirements. It contains statements and diagrams to communicate the final solution.

Several diagrams that show how the (sub-)system addressed by the pattern is designed, must be provided. They must clearly capture and describe the following aspects:

- Which are the components that participate to the realization of the sub-system? In case a concrete implementation (e.g. a product) exists for some of them, the description of these implementations can also be added and reported in the section “Relationships to Components Catalogue”.
- How are the provided functionalities of the (sub-)system decomposed and allocated to the different participant components? In case functionalities are compliant to a specific standard, it is also important to specify such a standard and report it in section “Relationships to Standards Catalogue”.
- How are the components inter-connected and how do they inter-operate to realize the provided functionalities of the sub-system and, eventually, how do they access its required functionalities? If communication is compliant to some standard, it is important to specify such a standard and report it in section “Relationships to Standards Catalogue”.

Hereafter a diagram that shows how a pattern can be expressed by using UML is depicted.

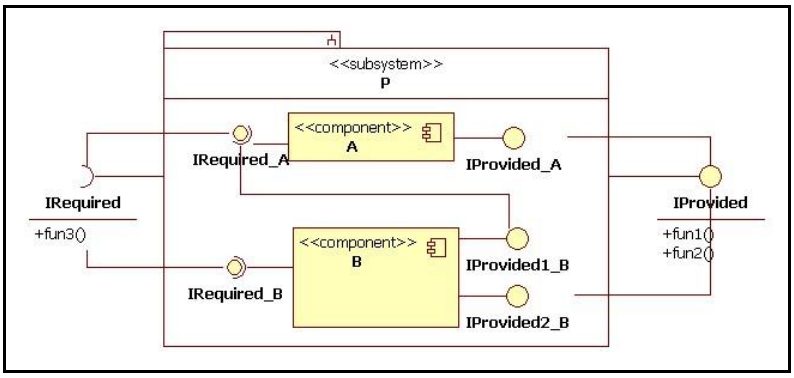


Figure 12: use UML to describe a pattern

The above diagram shows the **architectural choices** made by the pattern to design the “subsystem P”. It introduces the components and their provided and required interfaces. It also shows how the components are inter-connected. A description of how the different components collaborate to realize the provided functionalities should be also given.

Note that the functionalities of the interface **IProvided** must correspond to those addressed in the section “Functional requirements”.

The functionalities of the interface **IRequired** must correspond to those stated in the section “Assumptions”.

Summarizing this section must contain the following diagrams and information:

- A component diagram showing the pattern as a black box component and its provided and required interfaces.
- A component diagram showing which are the components of the (sub-)system and how the components are inter-connected.
- The description of each component of the (sub-)system and of its provided and required interfaces in a sub-section called **Components Descriptions**
- The description of how the different components collaborate to realize the provided functionalities of the overall (sub-)system in a sub-section called **Functionalities description**. Sequence diagrams must be designed for describing each provided functionalities of the (sub-)system.

4.5.7 Relationships to other patterns

This section is intended to provide the relationships that the proposed pattern has got with other patterns, so that they can be used to scale beyond point solutions, to address larger and more sophisticated problem spaces. The admitted relationships are the following:

- Extends relation;
- PartOf relation;
- ComplementsWith;
- CompetesWith;
- IsApplicableTo.

See section 4.3 for a detailed explanation of each relationship.

4.5.8 Relationships to Components Catalogue

This section is intended to provide references to the building blocks of the component catalogue [see section 2.3] of the NEXOF Reference Architecture, that can be used to implement some of the components, interfaces and connections of this pattern. This catalogue groups both abstract descriptions of components (e.g. an UDDI registry) as well as product or software-based (concrete) components (e.g. the jUDDI library).

However the patterns will be the main sources to the development of the component catalogue The catalogue will contain all the abstract and concrete components promoted by the patterns. Hence, until the first version of the catalogue will not be released, the pattern has to provide the complete description of the components it candidates to be part of the catalogue. This contribution is also required when the catalogue will be available but it does not contain the needed components.

This section is parted into two subsections, specifically dedicated to abstract and concrete components.

The **concrete components** section provides references to specific implementations and short descriptions of the main features of such implementations. It must also contain the relationship between each implementation and an abstract component of the catalogue.

The **abstract components** section introduces all the abstract descriptions of the concrete components. They are relevant only when they are related to one or more concrete components. Each abstract component is presented through a short description of its main features.

NOTE: each concrete component must correspond to an abstract one and this fact must be clearly stated.

4.5.9 Relationships to Standards Catalogue

This section is intended to provide references to the standards used to implement some of the components, interfaces and connections of the proposed pattern.

The standards listed in this section will contribute to the development of the standards catalogue [see section 2.3] of the NEXOF Reference Architecture.

4.5.10 Application examples

This part is optional. It is intended to provide examples of the pattern found in a real system.

4.5.11 References

This section is intended to provide list of initiatives, technologies and other kind of sources related to the pattern (not necessarily used in the pattern).

5 THE SPECIFICATION PROCESS

The *NEXOF-RA Project* has adopted a top-down production process to better support the production of a set of inter-related patterns. Starting from the production of top-level patterns, i.e. the most general and abstract patterns, other patterns are produced with respect to other already-committed patterns. This way, the problem they address and the context where they are applicable are clearly and well-defined. This approach makes the verification of the consistency of the overall set of patterns as well as the instantiation of concrete architectures easier and more controllable.

The next figure gives a very high-level description of the pattern development process.

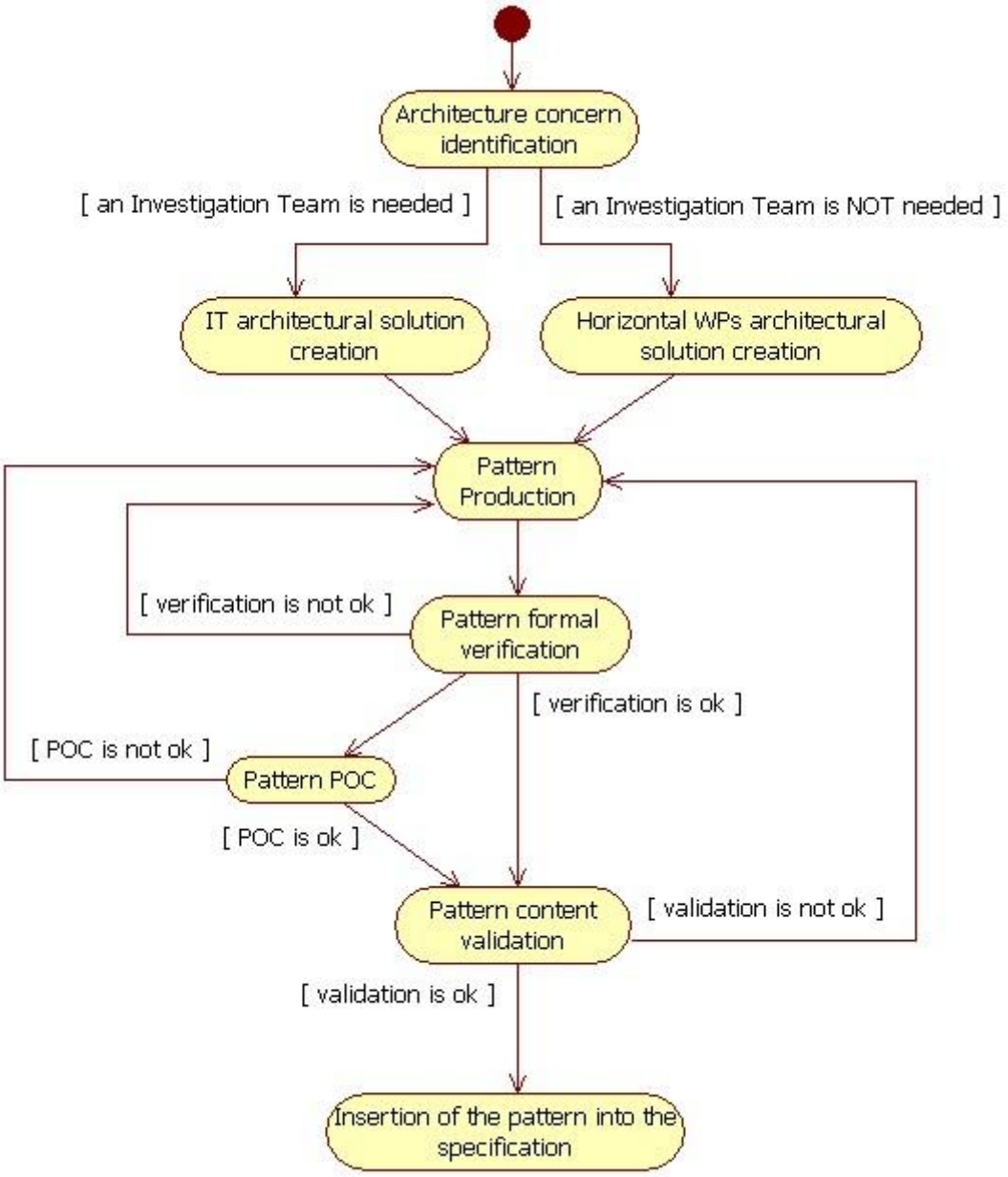


Figure 13: Patterns Development Process

When an architectural issue, related to SOA infrastructures, is identified, the *NEXOF-RA Project* expert people start working on the elaboration of new architectural patterns. This activity is performed within the research tracks of the Project and can be supported by means of external contributors (i.e. Investigation Teams). Contributors must use the pattern template document[5] to produce the pattern description document.

After new patterns have been produced, they are submitted to the specification development workgroup (WP7) to check their relationships to the other patterns

[5] It can be downloaded at http://www.nexof-ra.eu/?q=rep/term/12

of the specification, the compliancy to the pattern template and the consistency of functionality requirements and assumptions with respect to the stated relationships. This checking action is called “formal verification” though it is not based on any formal logic. It mainly deals with constraints on the format of the pattern description and takes it apart from the content validation that evaluates the quality of the pattern’s architectural choices. If the formal verification is not ok, WP7’s feedback is returned to team working on the elaboration of the pattern, where an update version of the pattern will be elaborated. This cycle can be repeated several times till the pattern succeeds WP7 verification.

Once WP7 verification is achieved, those patterns that address challenging architectural problems of SOA Infrastructure design are selected to be validated through proof-of-concept actions. In any case all the patterns are submitted to the *NEXOF-RA Architectural Board* (AB) for the validation of the content. In this phase, the pattern is evaluated with respect to its architectural design added-value, originality and innovation. If it is validated by the architectural board, the pattern will be added to the *NEXOF-RA Specification*, otherwise the pattern and the negative feedbacks will be returned back to the development team for further improvements.

To better explain the above process, next figure depicts all the various states a pattern can be during its development process.

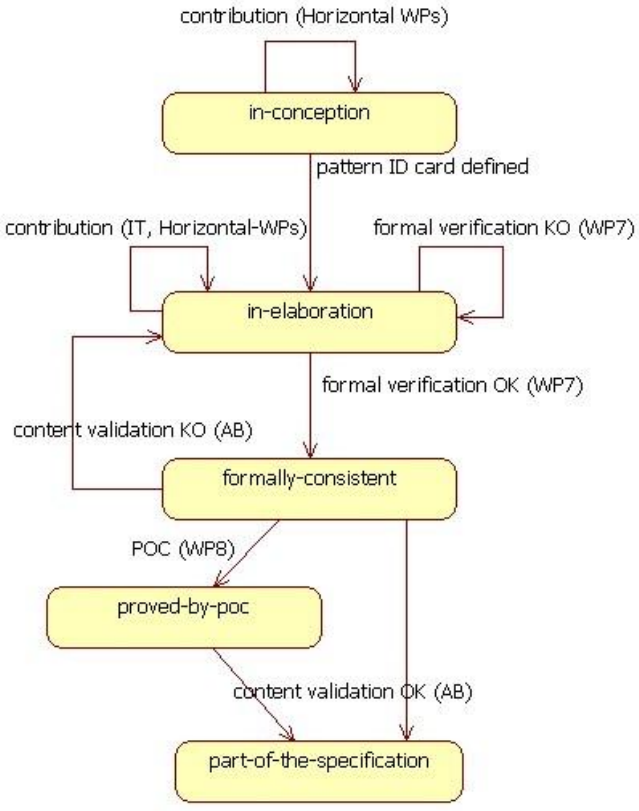


Figure 14: Pattern States

As shown by the previous figure, a pattern can be in one of the following states:

- **in-conception**: to refer to a pattern that has been identified and qualified only by means of the architectural problem it addresses. The architectural solution proposed by a pattern in this state has not been elaborated yet.
- **in-elaboration**: to refer to a pattern that is under development. A draft of the architectural solution proposed by a pattern in this state is available.
- **formally-consistent**: to refer to a pattern that has been already elaborated and has succeeded the formal verification.
- **proved-by-poc**: to refer to a pattern that has been successfully proved by a proof-of-concept action.
- **part-of-the-specification**: to refer to a pattern that has succeeded the formal verification and has received the final approbation of the *NEXOF-RA Architectural Board* to be part of the *NEXOF-RA Specification*.

REFERENCES

[1] [Buschmann et al. 1996] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, & M. Stal. Pattern-Oriented Software Architecture: A Pattern System. Addison-Wesley, Boston, 1996.

[2] Erich Gamma, Richard Helm, Ralph Johnson, and John M. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional, November 1994.

[3] Alexander, C.; Ishikawa, S.; Silverstein, M.; Jacobson, M.; Fiksdahl-King, I.; Angel, S. A Pattern Language. Oxford, University Press, New York, 1977.

[4] Buschmann, F.; Meunier, R.; Rohnert, H.; Sommerlad, P.; Stal, M. Pattern-Oriented Software Architecture. A System of Patterns. Volume 1. John Wiley & Sons Ltd, 1996.

[5] [Gamma et al. 1995] E. Gamma, R. Helm, R. Johnson, & J. Vlissides. Design Patterns: Elements of Object-Oriented Software. Addison-Wesley, Boston, 1995.

[6] [Riehle 1997] D. Riehle. "Composite Design Patterns." Proceedings of Object-Oriented Programming, Systems, Languages and Applications, OOPSLA '97, pp. 218–228, Atlanta, GA, October 1997.

[7] RA Model V2.0, http://www.nexof-ra.eu

[8] NEXOF-RA Glossary, http://www.nexof-ra.eu/?q=node/187

[9] Quality Model for NEXOF-RA Pattern Designing report, http://www.nexof-ra.eu

[10] Software Architecture in Practise. Bass L., Clemens P., Kazman R. 1998. Reading, MA: Addison-Wesley

[11] Toward Deriving Software Architectures From Quality Attributes, Rick Kazman, Len Bass, Rick Kazman, Thomas R. Miller, Lt Col, 1994

[12] Architectural Patterns Revisited – A Pattern. Language. Paris Avgeriou. Uwe Zdun. In Proceedings of the 10th European Conference on Pattern Languages of Programs (EuroPLoP 2005), 2005

[13] Sherif M. Yacoub; Hany H. Ammar Pattern-Oriented Analysis and Design: Composing Patterns to Design Software Systems, Addison-Wesley, 2003.

[14] Thomas Erl. http://www.soaprinciples.com/, 2007-2008