Politecnico di Milano
Facoltà di Ingegneria dell'Informazione

Tesi di Laurea in

Ingegneria Informatica

The Impact of SOA on Interoperability: a Systematic Literature Review

Supervisor
Luciano Baresi

Assistan Supervisor
Patricia Lago

Author
Marco Muffatti, 724742

Academic Year 2009/2010

Contents

List of Figures

List of Tables

Abstract.

Service-Oriented Architecture (SOA) is an emerging software approach to implement distributed systems. It is based on the integration of heterogeneous applications to deal with rapid changes in the business goals. Independent services are the components of the system. Loosely coupling and dynamic binding enable a much more flexible integration than existing solutions. However, adopting Service-Oriented Architectures has many implications on the system and its quality. Interoperability is a quality attribute that refers to the ability of two or more elements to exchange information and to use it. Starting from some differences between Service-Oriented and traditional software engineering we highlighted in previous studies, we want to analyze the impact of SOA on interoperability. Our methodology consists of a systematic literature review, which aim is to provide evidences relevant to the mentioned issue. We used as sources some digital libraries and we documented all decisions concerning selection criteria, so other reviews can be performed using this model. Review findings point out that differences between SOA and traditional approaches have different impacts on different interoperability levels (in particular syntactic, semantic and pragmatic). Our studies and considerations do not refer to specific applications or technologies; however, sometimes we introduced Web services examples to clarify practice concepts or real adopted solutions.

Prefazione.

Negli ultimi anni le architetture orientate ai servizi stanno avendo una diffusione sempre più ampia in ambito sia industriale sia accademico. Questo tipo di approccio si basa sull'integrazione di elementi, detti servizi, che permette di raggiungere gli obiettivi di business richiesti. Nonostante la loro natura molto eterogenea, i servizi possono essere composti in sistemi molti complessi, sfruttando il loro forte disaccoppiamento e la possibilità di effettuare binding dinamici a runtime. Un aspetto che ricopre un ruolo fondamentale all'interno di queste architetture è il livello di interoperabilità raggiunto tra i vari partecipanti. In questo report andremo ad analizzare l'impatto che ha un approccio orientato ai servizi su questo attributo di qualità attraverso un insieme di differenze, ricavate da alcuni studi precedenti, tra l'ingegneria del software tradizionale e quella dei sistemi Service-Oriented. Il nostro metodo è basato su una revisione letterale, eseguita in maniera sistematica per giungere ad una serie di evidenze riguardanti la nostra questione. Partendo da alcune librerie digitali abbiamo definito tutti passi necessari per il raggiungimento del nostro obiettivo nella maniera più formale possibile e abbiamo giustificato tutte le nostre scelte in modo da poter anche riutilizzare questa struttura come punto di riferimento per altri lavori simili a questo. I risultati della revisione ci hanno portato a sostenere che le differenze tra SOA e le architetture software tradizionali hanno un impatto diverso ad ogni livello dell'interoperabilità, in particolare in quello sintattico, semantico e pragmatico. Nonostante tutte le analisi effettuate abbiano una validità generica, abbiamo menzionato in alcuni casi le soluzioni adottate dai Web services, i quali rappresentano la tecnologia orientata ai servizi più diffusa oggigiorno. Ciò può essere utile per capire meglio alcune problematiche reali e la situazione attuale di alcuni aspetti pratici delle SOA.

Chapter 1

Introduction.

1.1 What is SOA.

A Service-Oriented Architecture is a set of services. These components can be invoked and their interface description can be published and discovered. Each service is the endpoint of a connection, which can be used to access the service and interconnect different services. Communication among services can involve only simple invocations and data passing, or complex coordinated activities of two or more services [1]. Basically, a Service-Oriented Architecture is very simple: a service is exposed through a remote interface (well-defined by an interface description that contains all the details about the service) then it advertise itself at a central lookup service, sometimes named broker. Client applications can find advertised services, by some properties, in the lookup service and they receive information about interaction details of how to communicate with the desired service.

The service provider offers a service to service clients, or consumers. Sometimes the service is not fully realized by the service provider implementation, but it can involve some backends, such as server applications, legacy systems, ERP systems and databases. Flexible integration of heterogeneous backends systems is one of the main goals of a SOA [1]. The use of backends in SOA is not strictly necessary but it is very important because they are involved many times in the service implementation. The service directory has a central role in the architecture. It contains references of services published there and clients can look them up in the registry. Object IDs are assigned to each service. These IDs are valid only in the context of a specific application and an absolute object reference solves this problem including location information, such as host name and port.
The interaction between service client and service provider is defined by ser-

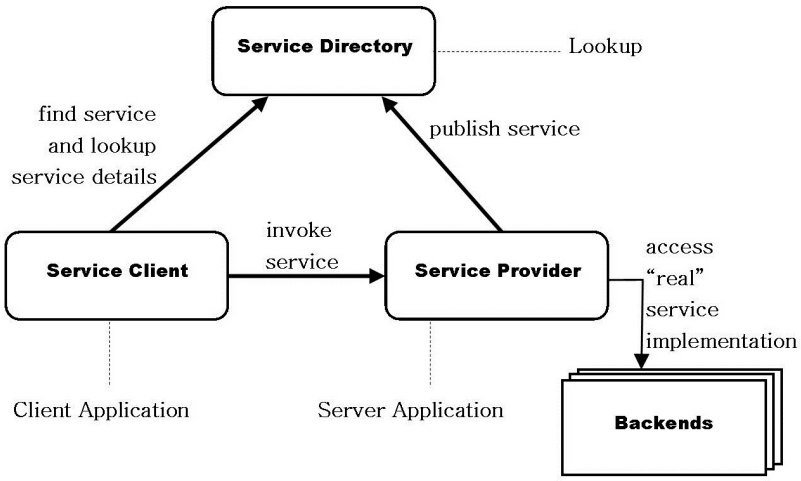

Fig. 1.1: SOA Architecture.

vice contracts. Design-by-contract approach is used as reference model because a service needs to be specified more than simple remote interactions, such as RPC-based invocations.
The service contract includes the following information about a service:

- communication protocols
- message types, operations, operation parameters, exceptions
- message formats, encodings, payload protocols
- pre- and post-conditions, sequencing requirements, side-effects, etc.
- operational behavior, legal obligations, service-level agreements, etc.
- directory service

Not all of these contract elements can be easily implemented. Communication and message properties are usually described with interface descriptions. The interface description of a Service-Oriented Architecture is more complex than those of a distributed object-oriented middleware because it has to describe a wide variety of message types, formats, encodings, payloads, protocols, etc. Lookup (or broker) is used to locate or obtain the interface description and the absolute object reference of a service. In addition, some of them provide other information about the service that is missing in the interface description, such as operational behavior, legal obligations and SLA. Furthermore,

specific-domain service properties or metadata of services exist.
A service contract can be realized with explicit and implicit contract specifications in electronic or non-electronic form. Implicit and non-electronic contract specifications are not very useful because a client would not be independent of provider features and this would contradict the principle of loose-coupling. Using explicit, electronic form and accessible at runtime contract elements, the broker can retrieve the best service provider for each consumer request.

To understand SOA it is useful to introduce the concept of interface and adapter. In fact, SOA is used within larger client and server applications and services are used only for integration purposes. Adding a service interface to the server application and a service adapter on the client side allows to isolate applications from changes in SOA (or in service contract). Using these

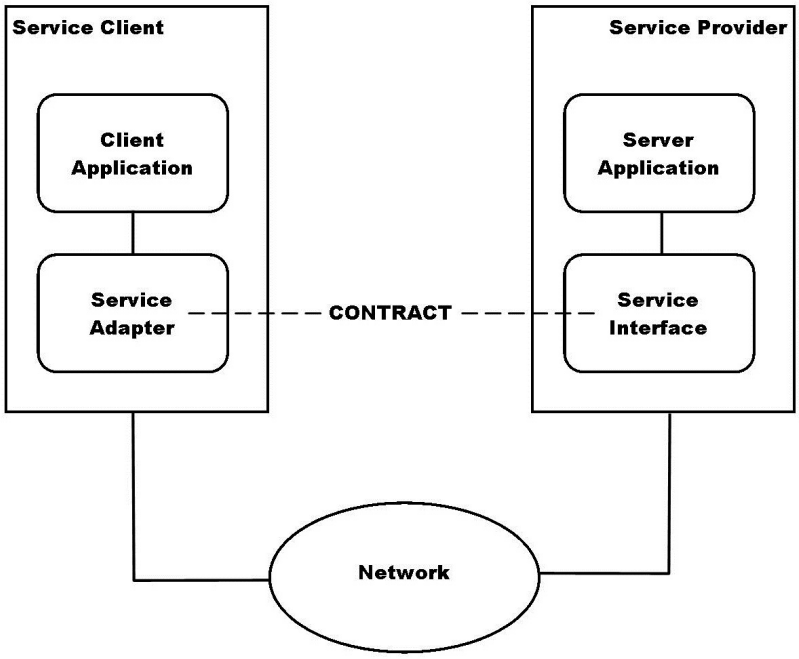

Fig. 1.2: Service Interface and Adapter.

components some problems arise because services sometimes are message-oriented, while other times they are RPC-oriented. As consequence different kinds of protocols are used, such as reliable messaging protocols or unreliable asynchronous RPC. Both client and server applications have to support many different service adapters and service interfaces.

We can identify four main layers in a Service-Oriented Architecture.

Usually this architecture is highly symmetrical on client side and provider side. In addition, orthogonal tasks (or aspects) implemented across these layers exist. Figure 1.3 shows them. Starting from the top, the main SOA

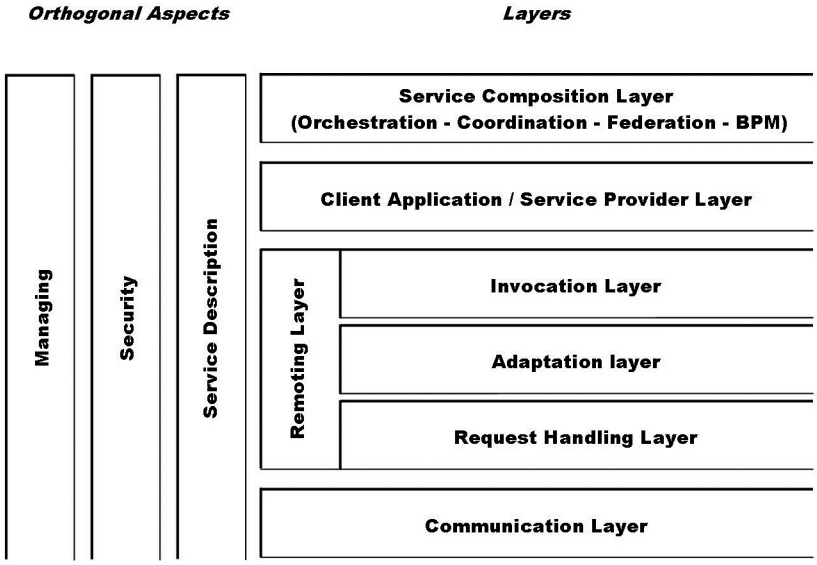

Fig. 1.3: Client and Server SOA Layers.

layers are:

- service composition. It includes all functionalities to compose services, such as implementations of service orchestration, service coordination, service federation or business process management. This layer is not mandatory and it is not always implemented.
- client application/service provider. This layer performs client invocations (service consumer side) and service implementations (service provider side).
- remoting. This layer implements the middleware functionalities of a SOA. It is composed by three sub-layers: invocation, adaption and request handling. These details of the consumer and provider are hidden because the broker hides and mediates all communication between the components of the system. The invocation layer is responsible for marshalling/demarshalling and multiplexing/demultiplexing of invocations/replies. The adaption layer adapts invocations and replies in the message flow. The request handling layer manages the basic tasks of establishing connections and message passing between consumer and provider.

- communication. The communication layer defines the message flow and it manages the operating system resources, such as connections, handles, or threads.

Vertical tasks involve all SOA layers. They are orthogonal aspects useful to realize every component of the architecture, such as security of services, descriptions, management functionalities for services, etc.

1.2 What is interoperability.

There are many definitions of interoperability because the term can have various interpretations in different contexts [2]. We can certainly assert that in software engineering context interoperability is a software quality attribute, a property of the system useful to evaluate the quality of service (QoS) provided by the overall solution.
IEEE has four definitions of interoperability [3]:

- the ability of two or more systems or elements to exchange information and to use the information that has been exchanged.
- the capability for units of equipment to work together to do useful functions.
- the capability, promoted but not guaranteed by joint conformance with a given set of standards, that enables heterogeneous equipment, generally built by various vendors, to work together in a network environment.
- the ability of two or more systems or components to exchange information in a heterogeneous network and use that information.

Summarizing we can say that interoperability is the possibility to let some different entities communicate inside a common environment, to understand and to use information exchanged between them. In particular, for our aims, interoperability is the ability of software and hardware on various machines from various vendors to communicate with each other without significant changes to either side [4].

Several interoperability models have been introduced in the literature to classify interoperability levels, such as Levels of Information System Interoperability (LISI) model or the NATO Model for Interoperability [2]. However these are technical models, too much related to the underlying technology or domain. Levels of Conceptual Interoperability Model (LCIM) was proposed

to deal with conceptual interoperability issues beyond technical interoperability [5]. We can use it as reference to classify different levels of interoperability. LCIM has seven levels, from "no interoperability", the lowest, to "conceptual interoperability", the highest, as illustrated in Figure 1.4.

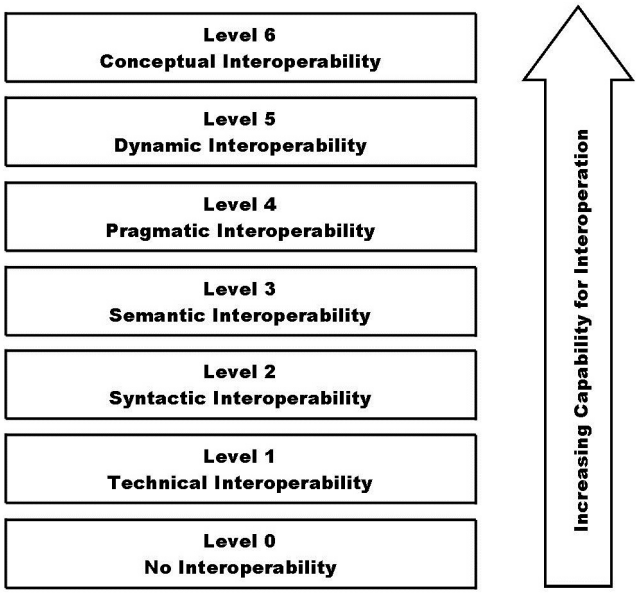

Fig. 1.4: Levels of Interoperability Model.

- level 0: stand-alone without connection systems have *no interoperability.*
- level 1: on this level a common communication protocol exists for exchanging data between participating systems. A communication infrastructure is established allowing it to exchange bits and bytes through underlying defined networks and protocols achieving a *technical interoperability.*
- level 2: the *syntactic interoperability* level relies in a common structure to exchange information, like unambiguously defined data formats.
- level 3: on *semantic interoperability* level the meaning of data is shared. A common information exchange reference model is used to reach this level.

- level 4: *pragmatic interoperability* is reached when the entities are aware of the methods and procedures that each other are employing. Systems have to understand the use of data; the context in which the information is exchanged is unambiguously defined.
- level 5: since data and operations change over the time the state of each system will be modified. *Dynamic interoperability* means that systems are able to comprehend the state change that occur in the assumptions and constraints that each other is making over the time, in particular in the effects of operations.
- level 6: *conceptual interoperability* is the highest level of interoperability. It requires that conceptual models will be documented based on engineering methods enabling other engineers to interpret and evaluate them. In other words, we need a fully specified but implementation independent model with assumptions, constraints, etc.

Actually, not all of these levels are strictly related to interoperability. In fact, level 1 is likely inherent to integratability, defined as the quality of an application that makes it easier for it to exchange data with another incompatible application. High levels, like level 5 and level 6, concern composability (the principle that deals with the capability of software components, or modules, to be combinated and assembled in various combinations to meet user requirements). If we exclude level 0 (no interoperability) we can focus on level 2, level 3 and level 4. These levels of interoperability are the most important in current software systems and many studies have been made about them because they are easier than dynamic and conceptual interoperability but more efficient than technical interoperability.
In addition, we can introduce two different roles about interoperability: descriptive and prescriptive. The descriptive role of interoperability model allows to evaluate a real system and to estimate its interoperability level. The prescriptive role suggests the approaches and requirements that must be satisfied to reach a certain level of conceptual interoperability; it is a sort of interoperability guidance model to follow at design time.

1.3 State of the art of interoperability in SOA.

Service-Oriented Architecture is an approach focused on software development to build loosely coupled distributed applications using a collection of services. Loosely coupled property is a fundamental principle of this kind of architecture. It means that services interactions are neither hard coded

(like in Object Oriented Programming) nor specified at design time (like in Component Based Modeling). On the contrary services are defined out of any execution context and interact on the fly without any prior collaboration agreement [6]. Again, loosely coupled reduces both the complexity and the cost of resources, communication and management [7].

The problem of interoperability is old as the existence of software systems [8]. A first idea was to make enterprise applications interoperable via central databases. This approach failed because of semantic problems; not enough semantics could be covered in the database schema to understand the semantics of data. Then non-interoperable applications were created based on decentralized data management. Another attempt was to save the original vision of data independence by federated databases. For the same previous reason it failed because it was impossible to create a global understanding of data without referring to application semantics, let alone business semantics. Another school of interoperability has been concerned with standardizing system interfaces in a way that one system can call the other system. Some of them are RPC, CORBA, JEE and .NET. Here the problem of interoperability is only about technical level and it fundamentally relies on information hiding. It is very easy to call remote service with the parameter values that are completely non-sensical.
Today we can assert that technical interoperability is achieved. Most of service technologies rely on well-defined network protocols to exchange any data in every communication stage.

In this report we do not want to discuss SOA technologies because our concepts have to be valid for any architecture based in services. Anyway in this section we will consider some implementations of SOA to understand what interoperability means in practice and how it is realized in every level. The most well-known instantiations of the service-oriented computing paradigm are Web and Grid services, but there are also other types such as P2P (Peer-to-Peer) services, which are currently gaining importance [9].
In particular, Web services (a.k.a. WS) are the most used SOA instantiations. They promise universal interoperability and integration by establishing commonly agreed protocols for mutually understanding what a service offers and for delivering this functionality in an implementation independent way. Interoperability of legacy applications is also enabled facilitating a seamless integration between heterogeneous systems. Furthermore, new services can be created and dynamically published and discovered without disrupting the existing environment. Thus, WS technology provides a means of interoperating between different software applications running on a variety of platforms and frameworks.
The various WS standards and enabling technologies address technical level

interoperability. The common standards for WS description publication and invocation are WSDL, UDDI and SOAP. These XML-based protocols have become standards de facto for web services, especially WSDL (Web Service Definition Language) and SOAP (Simple Object Access Protocol) that provide the basis of service API description and service interoperation [10]. Web service technologies are evolving toward being able to support more advanced functionalities such as discovery, security, transactions, reliability, and collaborative processes management.
Several proposals have been made with some standards like UDDI (Universal Description, Discovery and Integration), WS-Security, WS-Transaction, WS-ReliableMessaging, BPEL4WS (Business Process Execution Language for Web Services) and WSCI (Web Service Choreography Interface). These standards constitute the basis on top of which developers can develop reliable and secure communication among services.
At the messaging layer, Web services use SOAP for document exchange and encapsulation of RPC-like interactions. However, the extensibility points provided in the specification are the source of interoperability issues. Instead, at the content layer WSDL describes Web services as collections of endpoints (port type) that describe the structure of the messages the service support [10].
Several standards, not only about interoperability, are present in the WS stack but usually only some of them are involved in a Web service implementation. In fact, more than one standard may cover the same aspect (for example WS-Reliability and WS-ReliableMessaging describe two different protocols concern Web service message reliability) or a standard cannot be used, such as WS-Security, WS-Management... Figure 1.5 shows a possible implementation of Web service stack.
Formally these standards and protocols are specifications. As mentioned specifications may complement, overlap and complete each other. They are usually referred as "WS-*". Nowadays more than 80 specifications exist but only few of them are largely used in implementations. Focusing on WSDL and SOAP we must specify that the first is not a protocol but a common language for defining Web service interfaces whereas the second is a protocol specification for exchanging structured information. Both rely on XML (extensible Markup Language) and the latter relies on application layer protocols, usually HTTP/S.
SOAP, WSDL and WS-Security are currently the most widely adopted and used specifications [11]. Although most platforms support UDDI, it is not widely used. Increasing interest in modeling service process and composition has led developers to support WS-BPEL.

All specifications are defined by organizations and each developer can

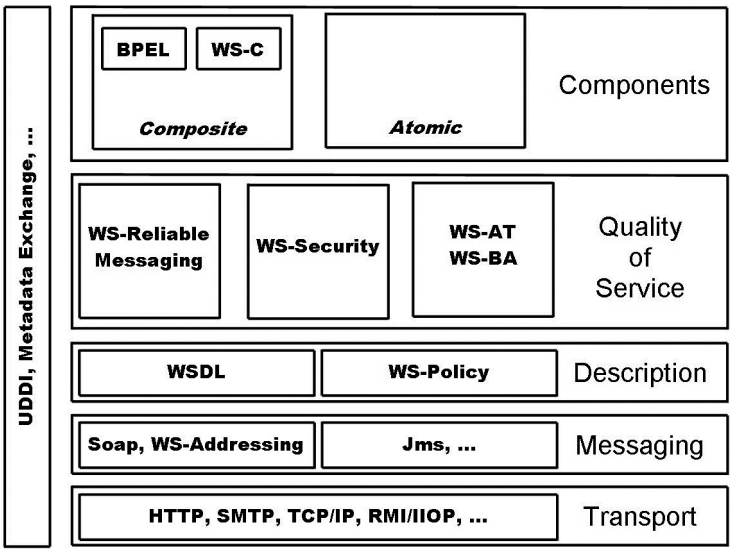

Fig. 1.5: Example of Web Service Stack.

choose which one to adopt. Web Services Interoperability Organizations, an industry consortium chartered to establish best practices for Web service interoperability, has defined specifications with regards to interoperability, named WS-I (Web Service Interoperability). It is not a protocol but a set of guidelines and tests. It relies on profiles that describe adherence to a group of specific versions of well-defined standards [12]. It is also their goal to provide tools and certify conformance with the profiles. Many Web service products were updated in recent years because of this initiative, in fact we can find commonly "compliant with WS-I Basic Profile 1.1" in data sheets nowadays. Unfortunately, WS-I has created a few profiles and other deliverables but a lot of work has to be done to cover all layers and standards in the Web service stack. Furthermore, WS-I profiles are not enough to satisfy all current industry requirements and they are only limited to elements of syntactic interoperability [13] and nothing has been made to address semantic issues. In fact, using XML interoperability among services developed by different organizations is guaranteed. Interoperability requires semantic as well as syntactic agreements in order to succeed [7].

Semantic interoperability is concerned with ensuring that the exchanged information has the same meaning for both message sender and receiver [14]. The data in the message have a meaning only when interpreted in terms of the respective subject domain models. However, the message sender does not always know the subject domain model of the receiver. Depending on

its knowledge, the message sender makes assumptions about the subject domain model of the receiver and uses this assumed subject domain model to construct a message and to communicate it. Semantic interoperability problems arise when the message sender and receiver have a different conceptualization or use a different representation of the entity types, properties and values from their subject domains [14].
Some examples of conflict are:

- naming conflicts: the representation is used to designate different entities, entity types or properties, or, conversely, different representations are used to designate the same entity, entity type or property.
- generalization conflicts: the meaning of an entity type or a property is more general than the meaning of a corresponding entity type or property.
- aggregation conflicts: an entity type or a property partially overlaps a corresponding entity type or property.
- isomorphism conflicts: the same entity type or property is defined differently in different subject domain models.
- identification conflicts: the same entity is identified by different properties.
- entity-property conflicts: an entity type is modeled as a property.

A necessary condition for the semantic interoperability of two systems is the existence of a translation function that maps the entity types, properties and values of the subject model of the first system to the respective entity types, properties and values of the subject domain model of the second system [14].
Coming back to our example of Web services, we can find different specifications about Web services semantic interoperability. Most of them rely on ontology to represent information in a domain, in particular on OWL (Web Ontology Language), a family of knowledge representation languages for authoring ontologies. The most used Web service implementations of OWL are OWL-S, WSMO, METEOR-S and WSDL-S.
The first one consists of three interrelated subontologies, known as profile, process model and grounding [10]. The profile describes the capabilities and parameters of the service. The process model details both the central structure and the dataflow structure required to execute a service. The grounding specifies the details of how to access the service (communication protocol, message formats, addressing, etc).

WSMO(Web Service Modeling Ontology) provides a conceptual framework and a formal language for semantically describing all relevant aspects of Web services in order to facilitate the automation of discovering, combining and invoking electronic services over the Web. It identifies four top-level elements as the main concepts for describing several aspects of semantic Web services [6]:

- ontologies. All resource descriptions as well as all data interchanged during service usage are based in ontologies. The core elements of an ontology are concepts (the basic entities of the agreed technology), relations (model interdependencies between several concepts), instances and axioms (define complex logical relations between the other elements defined in the ontologies).

- WSMO Service description. A service capability describes the provided functionality. A capability is described in terms of preconditions, assumptions, postconditions and effects. A service interface describes the behavioral aspects of a service in terms of choreography (how they interact with the service to consume its functionality) and orchestration (how service functionality is achieved by aggregating other Web services).

- goal. We can define WSMO goal as an high level description of a task required to be solved by Web services. Similar to a WSMO service, a goal consists of non-functional properties, a capability describing the user objective and an interface reflecting the user behavior requirements.

- mediation. WSMO ensures dynamic interoperability by defining mediators during design time that will be used by mediation components during runtime to resolve heterogeneity on the fly. A mediation tries to resolve mismatching that may arise between different used technologies (data level), or interaction protocols (process level).

METEOR-S project addresses the usage of semantics to support the complete lifecycle of Semantic Web processes using four kinds of semantics: data, functional, non-functional and execution semantics [6]. The data semantics describe the data (inputs/outputs) of the Web services. The functional semantics describe the functionality of a Web service (what it does). The non-functional semantics describe the non-functional aspects (like Quality of Service and business rules). The execution semantics model the behavior of Web services and processes. METEOR-S does not define a fully conceptual

model for Semantic Web services description but it follows a light-weight approach by extending WSDL files with semantic annotation. The semantic annotation is achieved by mapping WSDL elements to ontological concepts. Specification for WSDL is named WSDL-S, a W3C member submission that provides a mechanism to annotate the capabilities and requirements of Web services (described using WSDL) with semantic concepts defined in an external domain model [15]. Externalizing the models allows WSDL-S to take an agnostic view towards semantic representation languages. This allows developers to build domain models in any preferred language or reuse existing domain models.

The third level of interoperability we highlighted in previous sections is pragmatic interoperability. Semantic interoperability adds meaning to syntactic elements but semantics do not use it. Pragmatic interoperability is concerned with ensuring that message sender and receiver share the same expectation about the effect of the exchanged messages [14]. When a system receives a message it changes its state, sends a message back to the environment or both. In most cases, messages sent to the system change or request the system state, and messages sent from the system change or request the state of the environment. That is, the messages are always sent with some intention for achieving some desired effect. In most of the cases the effect is realized not only by a single message, but by a number of messages sent in some order. Pragmatic interoperability problems arise when the intended effect differs from the actual effect.
A necessary condition for pragmatic interoperability of a single interaction is that at least one result that satisfies the constraints of all contributing systems can be established. Being a service a set of related interactions between the system and its environment, *another necessary condition for pragmatic interoperability of a service is that the previous condition is met for all of its interactions and they can occur in a causal order, allowed by all participating systems.* Pragmatic interoperability can only be achieved if systems are also syntactically and semantically interoperable [15].

About Web services specifications able to ensure pragmatic interoperability do not exist. Some models have been proposed but none of them has been implemented in practice for Web services. These models usually rely on mapping information (like in semantic interoperability) and system behavior, and definition of conditions and constraints of information exchanged between services.

1.4 Motivation of the work.

SOA is a rather recent architectural pattern. Many studies have been made in last years to formalize this new way of developing software but there are a lot of aspects that have not yet been discussed or have been partially considered. Moreover, most of the implementations of SOA have been developed by industry, and then each vendor produced its own applications, following only some main guidelines. Just after the first applications, industries decided to define some useful standards to develop service-oriented components and to exchange information. Then, also other software organizations, like W3C, released their open standards, in particular for Web services, the most widely implementation of SOA. Starting from previously studies that tried to highlight differences between Traditional Software Engineering (TSE) and Service-Oriented System Engineering (SOSE), we want to study which is the impact that Service-Oriented Architectures have on interoperability. Indeed, a new architectural pattern not only affects the software design of the system, but also its quality attributes.

Quality attributes are non-functional requirements that describe the overall quality of the system. Achieving system goals for runtime quality attributes is critical for developers of applications that consume services for several reasons [13]. In fact application developers:

- need to be confident that the services (and compositions of them) will meet end user quality requirements.
- need to understand the cost and risk of achieving quality requirements, given that system qualities often must be traded off or built in.
- require information for selecting between alternate services with similar functional capability.
- require information about QoS (Quality of Service) to monitor and enforce Service Level Agreements (SLAs).

Dozens of quality attributes exist, such as reliability, availability, usability, security performance, interoperability... It is the latter that we are going to study and to analyze although some surveys have already been done.
Some researchers argue that through the use of the underlying standards, a SOA provides good interoperability, allowing services and applications built in different languages and deployed on different platforms to interact [15]. Moreover, they claim that the impact of SOA on interoperability is good because interoperability is the major benefit of services although semantic interoperability is not fully addressed and its supported standards are still

immature.
Actually, we think that a better study has to be done because sometimes studies concern Web services instead of SOA. As consequence, many conclusions may not be exactly related to this architectural pattern but specific to an implementation.

We decided to perform a systematic literature review, following the algorithm described in the next chapter. Results of this selection will be analyzed through differences between Traditional Software Engineering and Service-Oriented System Engineering. By doing this, we will obtain systematic results that will be the basis of our considerations of the impact of Service-Oriented Architecture on interoperability.
All criteria that we have chosen will be justified in the following chapters so that other systematic reviews can be easily made using ours by simply changing values of parameters.

Chapter 2

Systematic Literature Review.

2.1 Introduction.

This chapter describes how we realized the review. Firstly we will explain why we have decided to adopt a systematic literature review and our pilot study. Indeed, some guidelines derive from results of an initial work that allowed us to focus on main concepts for the review.

The second section instead shows the review protocol we followed, motivating all decisions we have taken about sources, strategies and selections. Results and their classification will be analyzed in next chapters.

2.1.1 Why.

We decided to perform a systematic literature review to analyze and discuss the impact of SOA on interoperability because this kind of approach will allow us to identify and select all high quality research evidence relevant to our issue. Systematic means that an explicit method has been performed to achieve results. We use some digital libraries as source and the first step has been the definition of the search strategy; then we have defined some criteria to filter these papers, selecting those relevant for our topic and excluding the others. The next step has been the data extraction, which consisted in the selection of all sentences in papers that contained keywords relevant for our subject. Subsequently, we have defined the criteria to select sentences that described the impact of SOA on interoperability and which difference (or differences) between traditional and service-oriented engineering they involved. Finally, we have grouped final sentences by topic because this kind of organization allows us to analyze all findings more easily.
Therefore the steps of our systematic literature review are:

- data sources: search engines selection.
- search strategy: query definition.
- study selection: inclusion/exclusion criteria.
- data extraction: selection and filtering of relevant data (sentences).

These procedures are very systematic and we can say that this is the most static part of the protocol. It is a sort of workflow that we have to follow in order to identify valid evidence. Obviously results will depend on values of parameters we are going to use. The choice of these values is the most variable part of the protocol and we cannot rely on automated methodologies to obtain the most appropriate values for our aims.

To solve this issue we did a pilot study that helped us understand the main topic, the most recurrent mistakes as well as paper compositions and keywords. We extracted from this study very useful information for the systematic literature review because we could focus more on our goals. Starting from it, we deduced many keywords for selections, some inclusion/exclusion criteria and how to classify final sentences. Next paragraph discusses about this preliminary study and its details and results are shown in Appendix B.

2.1.2 Pilot study.

When we started our work we did not know which was the actual state of research about quality attributes in Service-Oriented Architectures so we decided to do a study about SOA and quality attributes. Previous studies argue that differences between Traditional Software Engineering (TSE) and Service-Oriented System Engineering (SOSE) exist. In particular [16] it defines 7 differences between these two approaches. We want to use them to analyze the impact of Service-Oriented Architectures on interoperability, selecting sentences from papers that involve at least one of these differences. The pilot study has not been very systematic, indeed we have taken some papers regarding SOA not specific to interoperability. The choice of interoperability has been made later. Initially, we wanted to understand the impact of SOA on quality attributes through the differences highlighted in previous studies. Since many quality attributes exist we decided to split the work for every attribute. The choice of interoperability derives from the high quantity of discussions about it in SOA, in particular in Web services applications. Furthermore many aspects of it are discussed in literature now, being services a recent software system approach. Other interesting quality attributes are surely security, reliability, availability, scalability and performance because

they represent key points of this kind of architecture. A full research should also involve other attributes, such as usability, extensibility, adaptability, testability, auditability, modifiability, operability and deployability.
Obviously, understanding the impact of a new software pattern on quality attributes is very useful because it allows us to evaluate the quality of the overall system and, eventually, to modify the design or the implementation of our project. Furthermore, the use of differences helps us to take decisions about the adoption of SOA by organizations because many problems arise during the implementation and the integration of service-oriented solutions.
Now, we are going to describe the steps of our pilot study. Appendix B contains all details and results. Firstly, we took some generic papers about Service-Oriented Architectures, not specific of quality attributes. Initially, we wanted to create a table with differences between traditional and service-oriented systems and all main quality attributes. Then, reading the 12 papers, we should try to tick the cells representing an impact of a difference on a quality attribute. In this way a check for each attribute meant that SOA has an impact on this attribute through a difference (or more than one).
The table was structured as Table 2.1.

	Diff A	Diff B	Diff C	Diff D	Diff E	Diff F	Diff G
QA 1		x					
QA 2	x			x	x		
QA 3							

Table 2.1: Pilot Study Table A.

We have to read the example table as:

- SOA has impact on Quality Attribute 1. The Difference B is involved.
- SOA has impact on Quality Attribute 2 through Difference A, Difference D and Difference E.
- SOA hasn't impact on Quality Attribute 3.

The methodology to choose if a difference is involved on the impact of SOA on a quality attribute relied on reading the papers. Whenever we found a sentence relevant for differences and quality attributes we marked the related cell(s).
Unfortunately this approach hasn't been very useful because for most of the quality attributes all differences were involved in the impact of SOA. For this reason we decided to study only the impact of SOA on interoperability. As mentioned, this is a very important quality attribute in SOA and we will see

why.
Hence, we changed our strategy and we focused only on interoperability. The methodology was always the same: reading the papers and selecting the sentences. Differently from previous methodology we read papers one-by-one and for each we enumerated its sentences, relevant for our aim. Finally we have filled the table with the numbers of the sentences (the number format is: "paper number"."sentence number"). To reduce the number of sentences we grouped them by topic, then, when possible, we concatenated them (two sentences that belong to the same group and that fill the same and only cell can be concatenated). Grouping sentences we changed their references. A letter refers to the group and a progressive number to the sentence inside this group. Then each sentence contains a reference number to its paper. This kind of classification makes it easier to analyze the impact of SOA on interoperability because each consideration that we are going to write regards a topic of software engineering, so we can discuss more about it than about a single sentence. The result is something like that shown in Table 2.2.

	Diff A	Diff B	Diff C	Diff D	Diff E	Diff F	Diff G
Interoperability	A	B2	A1 D	B1	C	C1 D1	C2 D2

Table 2.2: Pilot Study Table B.

The letter without number means that all sentences of that group are in the cell, whereas a letter concatenated with a number refers to a single sentence.
We completed the pilot study writing our considerations for every element in the cells. The structure of considerations was:

- description of the topic in Traditional Software Engineering.
- description of the topic in Service-Oriented System Engineering.
- evaluation of the impact of this difference on interoperability.

Indeed, the first two descriptions define the difference and, as consequence, this difference has an impact on interoperability. The last part describes this impact. Obviously, considerations are partially subjective but for this preliminary study the main goal is the understanding of the argument. Then it is very useful to deduct keywords and key points for the systematic literature review.

2.2 Review protocol.

As mentioned, we performed a systematic literature review because it summarizes existing evidence, identifies gaps in current research and areas for further investigation, and provides a background in which to position new research objectives. The protocol we followed consists in four main phases, each one described in the next paragraphs. Our aim is to identify previous properties regarding the impact of SOA on interoperability, thus we used the pilot study to extract some information to define parameters, such as research strings or filter criteria.
We started choosing the sources of the review and then we created the queries to apply on them so that we obtained some hundreds of papers. Later we defined criteria to include or exclude papers relevant for our aim. Finally, we selected relevant sentences from these papers filtering those sentences containing keywords deducted from the pilot research.

2.2.1 Data sources.

The choice of data sources has been taken observing the current scientific world. Computer science field publications are contained in electronic libraries accessible via Internet. These resources are very useful because they allow fast and customized researches and all contents are always updated. Many digital libraries (or search engines) exist on the Web that contain publications such as papers, standards, e-books, digital subscriptions, conference publications, etc.
We selected three digital libraries we believe are the most important in the world on software engineering:

- IEEE Xplore Digital Library[1]
- ACM Digital Library[2]
- SpringerLink[3]

These databases are mostly used in doing systematic literature reviews and their search engines accept similar input query formats so we can define a generic query and adapt its syntax to all search engines. As consequence, results of one search engine will be coherent with the ones of the other. Moreover some papers will be contained in more than one database, thus some results will be overlapped.

[1] http://ieeexplore.ieee.org
[2] http://portal.acm.org
[3] http://www.springerlink.com

2.2.2 Search strategy.

This paragraph describes the queries we used in the search engines selected before. The title of our research is "The impact of SOA on interoperability" and the basic terms for the systematic review are *SOA* and *interoperability*, whereas *impact* is what we have to deduce and to prove (the evidence).
This first step of the protocol is divided in two phases to define research queries.
The first phase consists in the definition of synonyms of the terms to make the search as exhaustive as possible. With synonym we mean words that refer to the terms in the same topic, for example we consider *service-oriented computing* as synonym of *service-oriented architecture*. Obviously the choice of synonyms is very free, there are not constraints and it is very subjective. For these reasons the pilot study is very useful since it helped us understand how we can approximately a desired final result. Furthermore, after that study, we can deduce which are the common words used in sections of papers relevant for us.
In the query synonyms have to be in OR whereas the terms (SOA and interoperability) in AND. In our case we have:

```
("service-oriented"' OR "service-oriented architecture"' OR
"service-oriented computing" OR
"SOA" OR "service-engineering")

AND

("quality attributes" OR "interoperability")
```

It is easy to observe that word variations are not present; indeed, the search engines we used automatically manage singular, plural, masculine, feminine etc. Instead, we have to pay attention to quotation marks because search engines interpret them as the beginning and the end of a sentence. Excluding them some engines put words in OR and this mistake may cause unexpected results. Being a digital research, the OR enlarges the number of results, while the AND reduces it. The words we selected also depend on this property. For example, the use of the word "heterogeneous" as "synonym" of service-oriented architectures, in OR, produced thousands of irrelevant results and for this reason we excluded it from the query.
In the second phase we have to fill in the fields of the query. The most common fields in search engines are title, abstract, authors, full text, years,

etc. After some test queries we decided to focus only on *titles* and *abstract* because researches in full text produced too many results and almost all of them were not relevant for our work. We have not considered the other fields, like authors, years or codes because they are useful only to find specific papers, whereas we do not know which papers will be found.
Initially the query for our work was:

```
(Title:"service-oriented" OR Title:"service-oriented
architecture" OR Title:"service-oriented computing" OR
Title:"SOC" OR Title:"SOA" OR Title:"service engineering")

AND

("Abstract":"quality attributes" OR
"Abstract":"interoperability")
```

The syntax may appear strange because *Title* and *Abstract* are repeated many times and there is not a string like (Title:“service-oriented” OR “service-oriented architecture” OR “service-oriented computing” ORĚ) AND (“Abstract”:“quality attributes” OR “interoperability”). Indeed some search engines execute the OR between the rule on the left and the rule on the right and the lack of a field before a word (or words in quotation marks) is considered as if the field would be “Full text” and results of the research could be wrong. Furthermore, we adopted an interesting strategy to improve the quality of the results. We tried to put in OR the query with the same query with exchanged terms. In practice, we use the words of the Title in the initial query for the Abstract in the second part of the new query and vice versa obtaining:

```
((Title:"service-oriented" OR Title:"service-oriented
architecture" OR Title:"service-oriented computing" OR
Title:"SOC" OR Title:"SOA" OR Title:"service engineering")
AND
("Abstract":"quality attributes" OR
"Abstract":"interoperability"))

OR

((Abstract:"service-oriented" OR Abstract:"service-oriented
architecture" OR Abstract:"service-oriented computing" OR
Abstract:"SOC" OR Abstract:"SOA" OR
Abstract:"service engineering")
AND
("Title":"quality attributes" OR
"Title":"interoperability"))
```

This query has to be adapted to the syntax of each search engine because some fields may be different, for example the title may be represented as Title, Document Title or Ti. Finally, search engines allow filtering results through some criteria, in particular by year but we didn't use this filter because papers are quite recent (usually less than 10 years and most of them less than 5 years).

2.2.3 Study selection.

After the digital research we have to filter these papers (or studies, because there would also be other kinds of publications) defining inclusion and exclusion criteria. A study is selected only if it satisfy all the inclusion criteria and is removed if it fulfills any of the exclusion criteria. All criteria have to be pre-defined and the pilot study helped us to take our decisions. Now we are going to define our criteria and their motivations; Table 2.2.3 shows them.

The first inclusion criterion is quite easy to identify because obviously we deal with Service-Oriented approach and its properties. The E1 criterion excludes all studies that discuss other topics. In fact some words are included in other research fields because they can have many meanings. For example SOC means Service-Oriented Computing or System-on-Chip, and the latter

Inclusion Criteria	Exclusion Criteria
I1: Service-Oriented is the main topic. Motivation: we are interested in Service-Oriented challenges, in particular architectures, computations and engineering.	**E1: Study not about Service-Oriented.** Rationale: sometimes there are keywords with multiple meaning, for example SOC also means System-on-Chip. These papers are irrelevant for our work. **E2: SOA is a small part of the study.** Rationale: we want to focus on SOA. If the study concerns distributed systems or software engineering we have not sufficient information for our aim. **E3: Study is about a specific application.** Rationale: if the study is a case study or a specific application the impact of domain specific information cannot be generalized in SOA.
I2: Explicitly discuss interoperability. Motivation: we want to study the impact on the interoperability in the sense of software quality attribute.	**E4: Not about interoperability, but quality attributes in general.** Rationale: we do not want information about the quality of SOA in general; we are interested in studies specific of the interoperability and its challenges in SOA.
I3: SOA scientific papers. Motivation: a scientific study of SOA improves the quality of the research and the challenges are well considerate from the scientific world.	**E5: Study not scientific (workshop, conferences, meeting. . .).** Rationale: a non-scientific study does not include enough information about the topic and it cannot prove clearly its objectives.

Table 2.3: Inclusion and exclusion criteria.

is completely useless for our aim. The second exclusion criterion means that if a study is not specific to SOA it has to be removed. Indeed some publications treat with topics more generic that SOA, such as systems or software engineering. SOA is a paradigm of distributed systems and some papers discuss the main topic, retaining only a few parts to Service-Oriented Architectures. Unfortunately, these kinds of studies are too general because they cope with the problems of the principle argument, whereas SOA is considered secondary and we cannot obtain enough information. The last criterion that derives from I1 filters specific studies of a domain or application because they propose solutions that can be right only in those cases. Moreover, solutions are sometimes technology dependent (or with domain constraints) and we cannot apply them to general SOA issues.
The inclusion criterion I2 requires that a study essentially discuss interoperability. Indeed E4 excludes studies that refer only to quality attributes or other properties of Service-Oriented Architectures. We want to analyze the impact of SOA only on interoperability; other attributes are not relevant for our work.
The last inclusion criterion regards more the type of publications than their contents. We need scientific documents to identify evidences because non-scientific studies might not have high levels of quality and information sometimes is not enough. Usually we selected digital papers (pdf) and we excluded all other formats such as summaries of conferences, workshops and meetings. Therefore, we defined three inclusion criteria, one for *SOA*, one for *interoperability* and a third one for the *type* of study. Starting from these inclusion criteria we derived the exclusion ones and we explained the reasoning that we followed. All criteria must be defined a priori and then they have to be applied to all studies obtained from the digital research. Usually, applying criteria on paper title and abstract can be selective even if for some papers it has been necessary to read the full text. Finally we obtained the primary studies. The next paragraphs describe how we extracted data from them.

2.2.4 Data extraction.

At this point of the review we have some papers that are very relevant for our work. Now, we have to extract data that we will really use to identify evidences. Our strategy consists of four phases:

- creation of keywords list
- filtering of sentences containing at least one keyword
- definition of inclusion conditions

- selection of sentences that respect these conditions

The first and the third step are completely subjective; the second step is automated, whereas we have performed the fourth. For these reasons we have to focus on the first and on the third step (the fourth one is only an application of conditions as well as a selection of papers due to inclusion and exclusion criteria in the previous paragraph).
Here we can understand why the pilot study is fundamental; the keywords we selected derive from this study and some of them are not trivial because they are terms strictly involved in this research field. The box 2.2.4 contains all the keywords we used.

```
action          endpoint            qos
adapt           flexibility         quality
approach        functional          rele
back-end        heterogeneous       run-time
backend         impact              runtime
block           integrat            semantic
build           integration         service engineering
change          interaction         service level agreement
class           interface           sla
compare         interoperability    stakeholder
compliance      legacy system       standard
component       message             syntactic
compos          modul               third part
consum          multiple user       third-part
cross           object-oriented     traditional
desgin          OO                  vendor
develop         open
difference      organization
different       paradigm
disparate       pragmatic
dynami          provider
```

Immediately, it seems that most of the keywords are incorrect because they are non-sense. Instead, we used those because the second step is automated and it finds all sentences containing exactly the terms we passed in input. Therefore we did not use the plurals because the filter already selects the sentence because of the singular terms without the "s" (for example, a word was *block* hence we did not add *blocks*, indeed the first keyword covers both).

For the same reason we cut some words that contain other keywords, for example *adapter* is covered by *adapt*. Instead, other terms appear in different ways, such as *run-time* and *runtime*, *back-end* and *backend*, or they can be acronyms, like *sla* and *service level agreement*.
Furthermore we built a non-case sensitive filter, so we defined all words lowercase.
The second step, as mentioned, is automated and it is very fast and easy. We have taken all papers and then we have applied a filter that selected all sentences containing one or more keywords. Our tool is a linux script that receives in input a file for keywords and a directory with all papers. Then it parses all papers and it puts selected sentences on output. All input files are text files that we derived from pdf papers. Actually, we removed useless text, such as the text of summaries, tables and references. In addition, after some tests, we decided to remove also paragraphs that are irrelevant for our work (for example SOA descriptions produce always the same sentences) before filtering papers.
The third step defines which are the conditions that filtered sentences have to satisfy in order to identify evidences. Since our objective is to study the impact of SOA on interoperability we have chosen two criteria. To be selected a sentence has to:

- describe the impact of SOA on interoperability, and
- involve at least a difference between TSE and SOSE

The first criterion means that SOA has effects on interoperability, whereas the second shows whether this quality attribute changes in Service-Oriented Architectures. Sometimes a single sentence does not describe enough the impact, but we can understand it from the context.

Differences need a more detailed discussion. Our work relies on previous studies that treated differences between SOA and traditional software. Actually these differences have not yet been formalized, so some studies try to do it. As reference we use a paper titled "On Service-Oriented Architectural Concerns and Viewpoints" where the authors define **7 differences** between SOSE (Service-Oriented System Engineering) and TSE (Traditional Software Engineering) and, respectively, between SOA (Service-Oriented Architecture) and SA (Software Architecture). This study focuses on the real (fundamental) differences and their implications on SOSE and SOA and it creates a framework that shows the relationship between these differences and the service aspects. Now, we briefly describe the framework (Figure 2.1) and we the differences. The framework identifies 7 differences between service-oriented approaches and traditional ones:

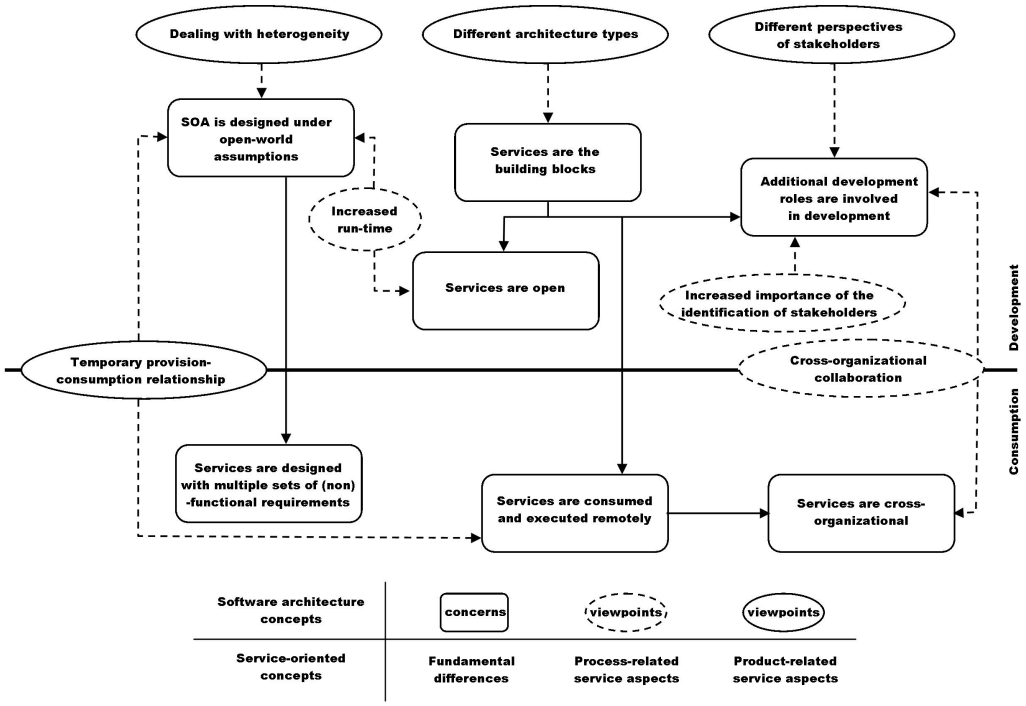

Fig. 2.1: A Framework of SOA Concerns and Service-Oriented Viewpoints.

- *Services are the building blocks*: instead of focusing on implementing a software system as a whole, SOSE and SOA focus on composing coarse-grained discoverable services acting as building blocks of service-oriented systems.

- *Services are open*: instead of not allowing any architectural changes after deployment, an architecture of a service-oriented system can be changed or even determined at runtime since services become the building blocks that can be composed at runtime.

- *Additional development roles are involved in development*: developer is not the only development role. This is rather split into three essential roles: service consumer, service provider and service broker since services are building blocks that need to be published and consumed.

- *Services are consumed and executed remotely*: instead of buying and installing software locally, users of services (pay and) consume services that are executed remotely at the service provider's side since services as building blocks are by definition used rather than owned.

- *Services are cross-organizational*: a software is often managed and owned by one organization. Instead, as services are not executed locally at the consumer's side, the control of services is often highly distributed and crosses trust and organizational boundaries.

- *SOA is designed under open-world assumptions*: instead of assuming stable execution environment, high uncertainty of the external environment has to be kept into consideration in SOSE and SOA.

- *Services are designed with multiple sets of (non)-functional requirements*: a software application is engineered for a single set of (non)-functional requirements. Instead, since the consumers as well as their needs are not completely known at design-time (according to open-world assumptions), services are engineered with multiple sets of (non)-functional requirements to fulfill different groups of potential consumers with different quality requirements.

Below we explain how the service aspects address the previous differences:

- *Increased importance of the identification of stakeholders.* The additional stakeholders involved in development increase the importance of capturing them explicitly in process models.

- *Cross-organizational collaboration.* Capturing the way in which cross-organizational collaboration is carried out is crucial to highlight that the additional development roles are distributed in different organizations and services are owned by different business partners.

- *Increased runtime effort.* The high uncertaintly resulted from open-world assumption demands for more runtime effort. Further, open services imply that more decisions have to be postponed at runtime. Making runtime effort explicit in a process model may highlight which uncertainty is dealt with and which decisions are postponed.

- *Different architecture types.* The architectures of services, composite services and service-oriented systems may indicate whether services are the building blocks of a service-oriented system.

- *Temporary provision-consumption relationship.* One of the open-world assumptions is not knowing the service consumers and their provider at design time. Further, the provision-consumption relationship in making SOA design decisions means making open-world assumptions explicit and highlights the fact that services do not execute locally at the service consumers.

- *Different perspectives of stakeholders.* A SOA design decision has different impact on different stakeholders. Explicitly capturing these impacts in making SOA design decisions points out the additional stakeholders involved in the development.

- *Dealing with heterogeneity.* Open-world assumptions suggest a heterogeneous environment where different data formats, protocols and technologies may coexist. Specifically, considering heterogeneity in the SOA design means making open-world assumption explicit.

The last step of data extraction is the selection of sentences that satisfy the two criteria defined in the third step. A sentence is selected if it describes one aspect of the impact of SOA on interoperability and it involves at least one of the 7 differences we explained above. A sentences that does not respect both criteria is excluded. As mentioned, a sentence can involve more than one difference, so we have to mark which ones they are, whereas the criterion of the impact is binary (if there is impact or not). To specify involved differences is useful for next stages, when we will write our comments and considerations. Furthermore, we are going to group selected sentences because in the pilot study we noted that some groups are very related with these 7 differences.

The next chapter shows results of the sistematic literature review and then we will make our considerations describing the differences between Service-Oriented and Traditional approaches.

Chapter 3

Review Results.

In this chapter we show the results of our systematic literature review. We proceed step by step, following the protocol defined in the previous chapter. We will add some figures and tables in order to make it easier to understand the workflow and how we obtained these results.

This work is divided in three parts; initially we discuss the results of the selection of the papers, then those of sentences and finally we will organize the sentences grouping them by topic.

3.1 Selected papers.

The digital research produced three lists of studies, one for each search engine: IEEE produced 197 studies, ACM 29 and Springer 78. Then, we read these studies to decide if they respected inclusion and exclusion criteria we defined in the protocol (actually we read the abstract, rarely the full text). Finally, we merged the results of the selections and we obtained 25 papers. Figure 3.1 shows all the numbers of this part of the work. Just to have some more information, we decided to merge the results after the selection. In the merging we found two papers that derived from two different search engines. In addition two papers were the same but they had different title. We considered these two papers as only one.
Here we number the 25 papers, so we will refer to them easily with an Id rather than using the title.

1. Bingu Shim, Siho Choue, Suntae Kim, and Sooyong Park. *A design quality model for service-oriented architecture.* 15th Asia-Pacific Software Engineering Conference, 2008.

2. Tom Goovaerts, Bart De Win, and Wouter Joosen. *A fexible architec-*

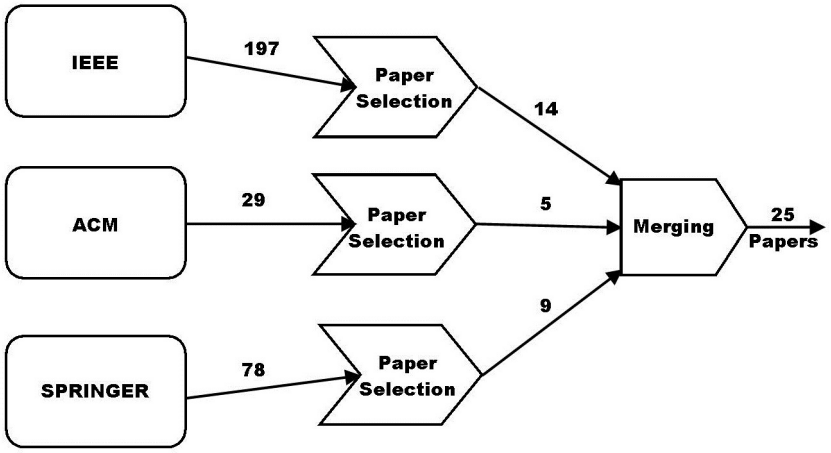

Fig. 3.1: Selection of the Papers.

ture for enforcing and composing policies in a service-oriented environment. International Federation for Information Processing, 2007.

3. Stanislav Pokraev, Dick Quartel, Maarten Steen, and Manfred Reichert. *A method for formal verification of service interoperability.* IEEE International Conference on Web Services (ICWS'06), 2006.

4. Liyi Zhang, Si Zhou, and Mingzhu Zhu. *A semantic service-oriented architecture for enterprise application integration.* Second International Symposium on Electronic Commerce and Security, 2009.

5. Leire Bastida, Alberto Berreteaga, and Inigo Cañadas. *Adopting service-oriented architectures made simple.* Enterprise Interoperability III, 2008.

6. Sujatha Kuppuraju, Aravind Kumar, and Geetha Presenna Kumari. *Case study to verify the interoperability of a service-oriented architecture stack.* IEEE International Conference on Services Computing (SCC2007), 2007.

7. Sriram Balasubramaniam, Grace Lewis, Ed Morris, Soumya Simanta, and Dennis Smith. *Challenges for assuring quality of service in a service-oriented environment.* PESOS'S09, 2009.

8. Grace Lewis, Edwin Morris, Soumya Simanta, and Lutz Wrage. *Common misconceptions about service-oriented architecture.* Sixth International IEEE Conference on Commercial-off-the-Shelf (COTS)-Based Software Systems (ICCBSS'07), 2007.

9. Giovanni Denaro, Mauro Pezzè, and Davide Tosi. *Ensuring interoperable service-oriented systems through engineered self-healing.* ESEC-FSE'S09, 2009.

10. Rodrigo Mantovaneli Pessoa, Eduardo Silva, Marten van Sinderen, Dick Quartel, and Luìs Ferreira Pires. *Enterprise interoperability with soa: a survey of service composition approaches.* Enterprise Distributed Object Computing Conference Workshops, 2008.

11. Balázs Simon, Zoltán László, Balázs Goldschmidt, Károly Kondorosi, and Péter Risztics. *Evaluation of ws- standards based interoperability of soa products for the hungarian e-government infrastructure.* Fourth International Conference on Digital Society, 2010.

12. George Athanasopoulos, Aphrodite Tsalgatidou, and Michael Pantazoglou. *Interoperability among heterogeneous services.* IEEE International Conference on Services Computing (SCC'06), 2006.

13. Aphrodite Tsalgatidou and Eleni Koutrouli. *Interoperability and eservices.* International Federation for Information Processing, 2005.

14. Marijn Janssen and Hans Jochen Scholl. *Interoperability for electronic governance.* ACM, 2007.

15. Sven De Labey and Eric Steegmans. *Making service-oriented java applications interoperable without compromising transparency.* Enterprise Interoperability III, 2008.

16. Jolita Ralyté, Per Backlund, Harald Kühn, and Manfred Jeusfeld. *Method chunks for interoperability.* Springer, 2006.

17. Liam O'Brien, Paulo Merson, and Len Bass. *Quality attributes for service-oriented architectures.* International Workshop on Systems Development in SOA Environments (SDSOA'07), 2007.

18. Stanislav Pokraev, Dick Quartel, Maarten Steen, and Manfred Reichert. *Requirements and method for assessment of service interoperability.* Springer, 2006.

19. Sami Bhiri, Walid Gaaloul, Mohsen Rouached, and Manfred Hauswirth. *Semantic web services for satisfying soa requirements.* International Federation for Information Processing, 2008.

20. Boualem Benatallah and Motahari Nezhad. *Service oriented computing opportunities and challenges.* Springer, 2005.

21. Jameela Al-Jaroodi, Nader Mohamed, and Junaid Aziz. *Service-oriented middleware trends and challenges.* Seventh International Conference on Information Technology, 2010.

22. Olaf Zimmermann, Vadim Doubrovski, Jonas Grundler, and Kerard Hogg. *Service-oriented architecture and business process choreography in an order management scenario.* ACM, 2005.

23. Jian Wang, Keqing He, Yangfan He, and Chong Wang. *Towards service-oriented semantic interoperability based on connecting ontologies.* International Conference on Interoperability for Enterprise Software and Applications China, 2009.

24. Andreas Winter and Jürgen Eber. *Using metamodels in service interoperability.* 13th IEEE International Workshop on Software Technology and Engineering Practice (STEP'05), 2005.

25. Hamid Motahari Nezhad, Boualem Benatallah, Fabio Casati, and Farouk Toumani. *Web services interoperability specifications.* Computer (IEEE Computer Society), 2006.

The figure 3.2 represents the time distribution of the papers over the years. We can say that all these papers are very recent because they have been

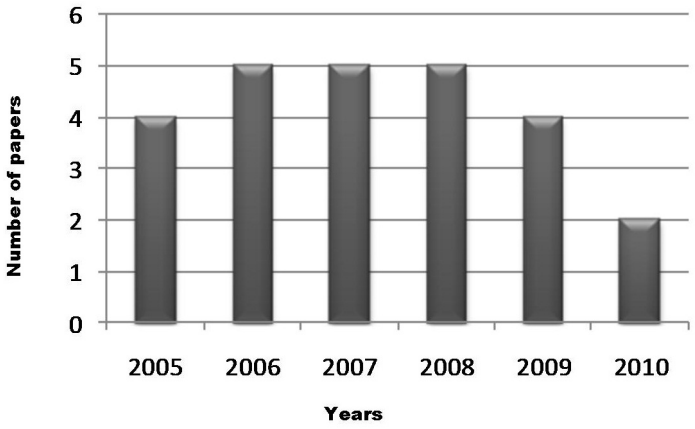

Fig. 3.2: Distribution of Selected Papers by Year.

published in the last 5 years.

3.2 Selected sentences.

Now, we have 25 papers and we have to filter all their sentences that contain the keywords defined in the section 2.2.4. As shown in the figure 3.3 we filtered 2042 sentences on which we made our selection. Finally we obtained 174 sentences, the core of our work. The table 3.1 indicates how many

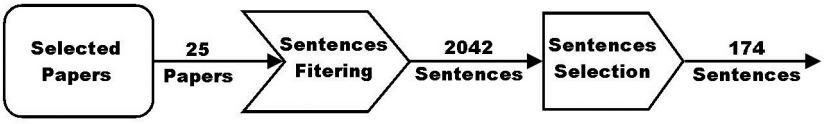

Fig. 3.3: Selection of the Sentences.

filtered sentences were contained in each paper, whereas the figure 3.4 shows the distribution of the final sentences.

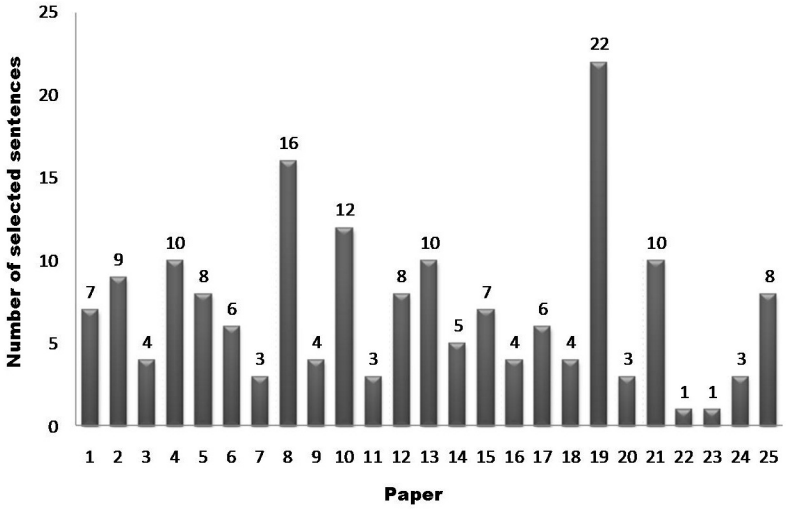

Fig. 3.4: Distribution of Selected Sentences for Each Paper.

3.3 Classification of final sentences.

The classification of the 174 sentences could be made in at least three ways:

- by paper
- by difference(s)

Paper Id	N. of Sentences
1	46
2	117
3	75
4	62
5	72
6	31
7	30
8	169
9	96
10	84
11	58
12	97
13	69
14	49
15	49
16	73
17	54
18	57
19	213
20	60
21	85
22	119
23	63
24	74
25	140

Table 3.1: Number of Filtered Sentences for Each Selected Paper.

- by topic

The first and the second one are useful to organize the sentences because if we know the paper or the difference, we can quickly find all related sentences. However, our work consists also in the analysis of the impact of SOA and the differences are means to achieve our conclusions. Hence, we decided to group the sentences by group, in order to discuss many aspects of SOA and, in this way, the considerations of the impact on interoperability will be more complete and detailed.
These are the topics we used to group the sentences (in alphabetical order):

- building blocks

- component modeling
- composition
- different service consumers
- dynamic business process
- heterogeneous components in the system
- interfaces and adapters
- many organizations in the system
- open world
- policies
- services are located differently
- system quality

Appendix A contains all topic tables. Each table is composed by three columns: paper number, involved differences and the content of the sentence. These tables are the last step of the systematic literature review and they are the starting point of our considerations that will be explained in next chapter.

Chapter 4

Findings.

4.1 Introduction.

Now, we are going to analyze final sentences with our considerations. We obtained 12 discussion topics about SOA's aspects. We have seen that some topics are closely related to some of the seven differences deduced in previous studies. Thus, we can define two macro categories for our discussion topics, those that represent a difference or not. Below we will show them in a systematic way. Each topic consideration will be composed by four parts:

- brief generic introduction to the topic
- considerations concerning traditional software approaches
- considerations concerning Service-Oriented Architectures
- impact on interoperability

We defined this structure to use it as a sort of guideline, whereas chapter 5 will contain last evaluations of the work.

4.2 Considerations on the SOA topics.

4.2.1 Building blocks.

Any software architecture is composed by elements (or components) and their relationships. Recursively each component can be decomposed into parts, relationships and constraints.

In traditional software engineering architects usually define more than one view of the project. The main organization of a system consists of the definition of macro components (and their relationships) then other views describe

these components in detail, how they interact between the environment and which constraints are applied. For example in client-server architecture the main components are clients and the server. If we focus on the server, we can design it as a three-tier architecture, with a view, a logical and a data layer. Each layer can be represented as a composition of objects and so on.

Service-Oriented Architectures are always composed by services that are the fundamental units of the composition. Services are loosely coupled and they do not depend on the other ones presented. This property allows to compose the system easily, dynamically and quickly; each element can contain the service description and the actual implementation. These concrete elements and their interactions are the architecture of the system [17]. SOA in fact, does not provide a complete architecture because it is an architectural pattern or a new way of developing software, but it is not the system architecture by itself [7]. Some implementations of SOA are Web services, Grid and P2P services (sometimes named eServices). The building blocks can be a service or a node on the Web.

The interoperability is a crucial aspect in SOA because it leverages the communication level between the blocks and the quality of the system composition. Since the elements are loosely coupled the communication is performed via standard protocols and adapters and interfaces allow the interaction with backends of service providers. Backends are usually legacy systems already running. Their integration within SOA is a process that may not be always easy and automatic [7] since this migration can imply a great adaption effort and the quality of this work can compromise the interoperability level of the component and, as consequence, of the whole system.

4.2.2 Component modeling.

At design-time software designers project the system and decide where to collocate all the components. In software engineering several patterns and approaches exist. A good designer has to be able to adapt the best model in order to optimize the costs of the overall project. Grain and detail of the design depend on various factors, such as the size of the project, the skills of the programmers or the level of the language used.

In traditional software the most used paradigm is Object-Oriented and, as consequence, many models have been developed for it. As a matter of fact, the family of UML (Unified Modeling Language) diagrams represents the starting point of the Object-Oriented modeling. In addition other approaches can be considered, for example the MVC (Model-View-Controller) is a widespread architectural pattern that split up the software components in three categories: the model defines the methods that interact with data,

the view shows data and interacts with users and the controller dialogues between these two components (a.k.a. business logic). This pattern allows each programmer to develop always the same very specialized type of code.

In SOA there are essentially three participants in the system: service providers, service consumers and the broker. When defining a Service-Oriented Architecture, the architect has to select which services will compose the system (orchestration) and how they have to interact (choreography). For these reasons services are considered the building blocks of Service-Oriented systems. In addition, a SOA is a distributed system and the components are located in different places, so the communication channel is always a network (usually Internet). However, most of service providers are not built from scratch, but they are legacy systems that have to be adapted through interfaces for this kind of communication.

Obviously, the level of interoperability is very relevant to allow the communication between provider, consumer and broker. Some standards guarantee a good level of syntactic interoperability, like WSDL, UDDI and SOAP for Web services, but the loosely coupled principle simplifies dynamic composition of the system on the one hand, and it complicates the communication between components on the other hand.

4.2.3 Composition.

A system composition is the aggregation of two or more elements in order to achieve a complex goal. On the other side, great systems can be decomposed in many components to subdivide the main process. Composition of systems includes both the selection of the components and the definition of their relationships and interactions.

In traditional software engineering the system composition concerns with the combination of architectural elements at design-time. Object-Oriented programming relies on models whose elements (classes) represent real elements and their interactions. Object-Oriented paradigm goal is the reusability of the components; therefore, they have to be defined as good as possible, in order to make their compositions easy when reused. In addition, the composition in UML is used to represent a class that contains other ones (that it is not to be confused with the aggregation). For instance, a car can be composed by an engine, 4 wheels, etc. Closed assumptions cause traditional components to be very static as well as their bindings, which are already defined before deployment.

Service composition is a fundamental mechanism in SOA that allows to combine the capabilities provided by several available services in order to satisfy more demanding needs [18] or to rapidly adopt to business changes.

In fact, SOA is an approach focused on software development to build loosely coupled applications using a collection of services [7]. The aggregation of services into executable workflow is called orchestration. The main advantage of SOA composition is that it can occur at runtime. This improves service dynamism, therefore if there are several feasible providers, the service consumer can choose, while it is running, which one to use [17]. The aim of composition is to use existing services to achieve some functionality that typically is not provided by a single available service or that has to be implemented from scratch. Actually, it is not easy to compose services dynamically. Currently, binding to service is usually done at design-time because the developer can discover the syntax and semantics before it is used [17].

Service composition plays a major role in enterprise interoperability [18] since its level implies which services can be added to the system. Semantic interoperability is the most open issue in service composition, whereas standards description languages such as XML guarantee syntactic interoperability for service interactions [19]. These standards simplify composition but dynamic binding and discovery at runtime require the use of ontologies and semantics to describe function and usage of services [17] [6].

4.2.4 Different service consumers.

A system consists of a set of applications that interact and exchange data. Essentially, two types of entities exist during an interaction: one of them requires information and the other one responds to the request. Depending on the context, these elements can be named differently, for instance client and server, or consumer and provider, and their roles can depend on the adopted paradigm, such as publish/subscribe, or architecture (e.g. P2P). Anyway, there is always one component that provides information and, at least, another one that use this information.

Traditional engineering systems are designed under closed world assumption, elements that compose the architecture are exactly known and during a consumption it is already established which is the provider and the consumer. Often the consumer is not specifically known, but the server can provide functionalities that a family of clients can require, for instance a Web server may not know precisely which clients will contact it, but it will respond to all HTTP requests received.

Instead, Service based Systems are characterized by being accessible by different and multiple users [7]. A service can be consumed by users or other services and they can require different information. Human users can consume both formal and informal descriptions since service description often contains an information part specified in natural language, whereas applica-

tions can use service descriptions to support interactions among various ways [11], for example, to verify that messages are exchanged in accordance with the defined protocol. In general, applications can only consume formal specifications. Since services are designed under open-world assumption, service providers do not known service consumers, so they have to support many clients with different requirements. Designing a service in a way that it can be used easily in many contexts is a much border task. If a service will be used in more than one context, it is also necessary to incorporate the requirements of many potential users and the usage patterns in the service design. Moreover, the service implementation itself should be architected so that it guarantees certain qualities desired by each of the stakeholders of the service [17].

Supporting many different consumers can cause many interoperability problems because the level of heterogeneity of the system increases. For example services have to support different versions of standards and specifications [4] since open world architectures are very dynamic and requirement changes are frequent.

4.2.5 Dynamic business process.

Organizations use IT to support their processes. These processes are dynamic because they follow the changes and strategies of the company. Changes are usually very fast and the software system has to adapt to them. There are different reasons that cause a business change, like:

- merge and acquisitions
- legacy systems integrations
- growing set of business rules and regulations

Furthermore, a process can involve other organizations (suppliers, third-parties, etc.) to realize the final product and a single change inside a single point of the system can have consequences on other companies. Hence, in general, changing a process is never trivial and the architecture model adapted is a key to manage the software structure of a company.

In traditional software approaches, every change of the business process means a change in the correspondent module (that can involve changes of many components). Usually, this kind of operation is very complex and it can be harmful for the component and for the whole system. In fact, it needs to re-design, re-develop and re-deploy the application. All these operations are very expensive in terms of time and costs but they are necessary because

bindings among components are always static, decided at design-time. There are some situations where the communication between some components results very complex , for example after an acquisition a company has to be able to integrate its systems with those of the acquired. Creating adapters can be the best solution but it is not very efficient because of their cost.

In SOA, instead, the composition of the system is completely different. Basically the application is dynamic and it can be adapted also at runtime. The main advantage is the possibility to interchange some services without interrupting the system. Hence, two or more components can switch from a protocol to another, change or add their APIs and interfaces and new services can be added inside the architecture without affecting the other ones. These features allow architects to adapt easily and quickly the application to the business process, exploiting its flexibility. Usually, discovery mechanisms and how to select a provider can be defined both at design-time and after deploy.

To let SOA's components communicate, all of them have to interact in a standard way. Many (XML-based) protocols have been developed to simplify their communication. Thus, the interoperability in SOA is better than in traditional architectures because there is a good mapping between non-standard service interfaces [20]. This allows new services to replace all old services, that are no longer useful, at runtime (because they are loosely coupled), at low-cost and dynamically. Web services, for example, are the most widespread dynamic structures [21] and their syntactic interoperability is guaranteed by three standard protocols (WSDL, UDDI and SOAP) that simplify the communication between all components available to accomplish business processes.

4.2.6 Heterogeneous components in the system.

Two or more applications are heterogeneous when they run on different platforms, have been written with different languages or have been designed by different vendors. Actually, the elements that are not compatible are messages, because heterogeneous applications can adopt different message formats and data cannot be exchanged or understood. Many problems can arise whenever a system tries to integrate new components, legacy systems or third-party products. Interoperability itself can be defined as the capability of multiple, autonomous and heterogeneous systems to use each other's functionalities.

Initially, even in traditional software, organizations could not exchange information because they operates widely disparate hardware that was incompatible [22]. Also Object-Oriented programming languages lack high-level support for platform-independent interactions, in Java, for instance,

the burden of guaranteeing sustainable interoperability is put entirely on the programmer [23]. Despite Component-based technologies have tried to address the issue of interoperability between heterogeneous applications, they have not provided a widespread solution for this problem [9].

The Service-Oriented paradigm builds on the notion of composing virtual components into complex behavior. Thus, a consumer can use the functionality offered by multiple providers without worrying about the underlying differences in hardware, operating systems, programming language, etc. To interact, services need not to share anything but a formal contract that describes each service and defines the terms of information exchange. Data are exchanged via messages conformed to the information model and semantics. Often a transformation between the enterprise semantics and the internal information model of the service is required [6].

Interfaces and adapters play fundamental roles to provide interoperability. Syntactic level is achieved easily in SOA and this is a main reason to adopt Service-Oriented technologies as integration system. Semantic interoperability can be attained with ontology mappings, for instance, through metamodel-driven approaches [21].

4.2.7 Interfaces and adapters.

Interfaces and adapters are subcomponents of a system that work as translators between two or more components. They perform the same functions and represent the network access point of a software element, but they are dual within interactions. The data requester uses the adapter to understand the same language of the interface exposed by the accessible component, in order to exchange data.

Thus, interface hides implementation and in traditional software engineering the information hiding is a principle that avoids security problems because the interfaces protect the access to some pieces of code. Moreover, if the underlying implementation changes, users are not affected by these changes.

Service-Oriented components also hide their implementations through an interface, but the main difference with the previous approach is that the service consumer does not know anything about the implementation of the service provider. In fact, services are loosely coupled to be more flexible in the composition and faster in the business changes. Clients are not concerned with how the providers will execute their requests and the way a client (which can be another service) communicates with the service does not depend on the implementation of the service (for example it does not need to know what language the service is coded or what platform the service runs on) [6]. Thus,

a service consumer will know only the information and behavior models of the provider and all the information necessary to determine if that service is appropriate for its aims. In addition all changes in service implementations do not impact on the service interface. Instead, the main goal of adapters is to translate client applications messages in order to be compatible with the language of the interfaces. Translations consist of mappings between message languages.

Thus, interfaces and adapters are elements that improve interoperability. Usually they rely on standard protocols and specifications. Indeed, they guarantee syntactic interoperability and this is the main reason why many organizations migrated to SOA. For example Web services rely on an open standard to expose their interface, WSDL (Web Services Description Language) and every programming language can define WSDL bindings that translate their method calls back and forth to WSDL [23]. Moreover, WS-* specifications are very flexible about interface definitions [11].

4.2.8 Many organizations in the system.

When a system is developed, its components can be local or remote whether they are running inside or outside the same environment. In particular remote components are located in other places, but inside organization boundaries. Boundaries refer to the logical location of the components because a system can be closed, but at the same time distributed.

In traditional software approaches, systems stay inside the same organization because all elements are developed for the enterprise business process and each one has its own specific functionalities. These architectures do not deny that using more than one product, language platform or subsystem. Moreover, inside the organization all components are known and the use of external elements is not a good choice because ad-hoc adapters have to be developed.

Instead, SOA is very dynamic and to rapidly compose systems it exploits the properties of all available services. These services can be either inside or outside the boundaries of the organization, since services are decoupled and their integration is very flexible [6]. For example a system can use services of many organizations because each service implement an application that only its organization can perform (because it only has necessary data or the algorithm is better than another).

Obviously, integration of heterogeneous applications leads to interoperability problems, In particular semantic interoperability is difficult to achieve because it relies on ontology. Ontologies are currently a major technology for supporting service description and composition but different organiza-

tions define ontologies in different ways [18] and even if some approaches define manual mappings or mediation techniques, good solutions for realistic applications do not exist nowadays.

4.2.9 Open world.

Open world assumption means that the environment of the system is not stable and can change frequently. On the contrary closed world assumption refers to unchanging and known environment. Actually, closed approach assumes that the external world changes slowly and the software can remain stable for a long period [24]. Moreover, the software itself is closed since it is composed of parts that do not change at runtime. Instead, in an open world the environment changes continuously and the system has to adapt and react dynamically. In addition, new components could be available in the environment and the system can discover and bind them dynamically to the application while it is running.

Traditional software systems are essentially designed under closed world assumptions because designers know which are the elements of the architecture and how they communicate. Object-Oriented programming is hard-coded and its components depend on the others, so each change can leverage many elements and, as consequence, it can cause many side effects. Instead, Component-based modeling relies on (third-party) components and it represents a step toward open worlds, but interactions between components are specified at design-time [6].

On the contrary, services are defined out of any execution context and interact on the fly without prior collaboration agreement. Each service should describe its (non-) functional characteristics the client has to understand if the service fits its needs. Moreover, services support dynamic binding that can occur at runtime because the integration of components in SOA is very flexible and fast. The use of a specific service can be chosen after deployment and, for instance, if a service fails or is unavailable, dynamic binding allows to replace it without interrupting the system. Service providers do not know service clients; therefore dynamic binding is guarantee by loosely coupling.

For these reasons we can argue that Service Oriented Architectures are designed under open world assumptions and interoperability between services is a critical factor for dynamic binding. Since services describe, through interfaces, their functionalities, clients have to be able to understand these information. Thus, semantic interoperability is required to improve flexibility and agility, in particular to compose systems at runtime. Domain ontologies help semantic protocols to focus their research inside a set of relevant solutions.

4.2.10 Policies.

Policies in software engineering can have many means. We want to focus on those concerning interoperability. Policies are rules that specify choices in the behavior of a system. By specifying them separately from the applications, the behavior can be changed dynamically by modifying the policy rules without affecting any application code [25].

Traditional systems do not need to specify many policies because applications are hard-coded and they know exactly how a component of the architecture communicate with the other ones. In addition, even if the specific element is unknown the iterations are well-defined (massage formats, protocols, etc.), but they are limited to the components of the systems.

Instead SOA relies on loosely coupled principle, so service environment can evolve and change without concern of its components. Furthermore, services are heterogeneous, developed in different languages and running on many platforms. Since service consumers may need specific requirements, which are very different from those of other consumers, service providers have to support more than one set of (non)-functional requirements.

Hence, policies are a way to define interactions between services. In particular, for interoperability a policy enforcement solution has to map different message formats with different types of requests for services [25]. The concept of generic message format is introduced; its role is like an intermediary during the interactions. The main advantage is that it is an adapter that translates back and forth between the generic format and the native ones [25], improving the interoperability between heterogeneous components. Every service can implement several policy-based adapters for different phases, such as authentication, business, etc.

4.2.11 Services are located differently.

Applications involving resources located in different places are called distributed systems. There are many advantages in adopting these architectures, in terms of costs, performance and development, moreover different paradigms exist to design them. Usually, a distributed system is composed by entities that communicate through a network and they interact in order to achieve a common goal; the problem is divided in tasks, or activities, each one performed by one, or more, entities.

Traditional software engineering relies on Object Oriented Programming, which focuses on combining elements of the domain problem in form of object containing data and methods useful to solve concerned problem and reusable for other ones. However, enterprise tired architectures evolved and demons-

trated that combining methods with data between tiers worked against scalability and loose coupling of the enterprise system, thus the use of data transfer objects between tiers and the focus on the data model for communication between tiers of the enterprise system [6]. In addition, these architectures and implementations did not provide a good solution for computing specialization and computing interdependence at business or government level.

SOA is a computing architecture that allows for complex relationships and specializations of computing services on a global scale. Services are available in a network such as Internet, but they can be developed also in-house. Making use of third-party external services improve the specialization of the services and the dynamism of the system because the application has not to be installed, but it is remotely consumed. This creates a marketplace, which is the functional area where service provider and potential service consumers meet and negotiate via a service broker to come to a formal agreement about consuming the services [7].

Standard communication protocols are needed for remote interactions. Since Internet is the most used network (but it is not the only one), communications between services rely on open standard protocols. For instance, Web services combine XML and HTTP to obtain SOAP and they use it to exchange data through the Internet. Standard protocols define common syntax and mechanisms of data exchange, in fact they are the main elements that allow achieving syntactic interoperability.

4.2.12 System Quality.

To evaluate the quality of a system usually metrics are used. A good design helps to achieve desired levels of quality. If a system is composed by many elements, the overall quality does not depend only on the quality of each element, but also on the type of their composition. The quality can be described through attributes, but some of them are not definable with metrics, such as testability, maintainability and usability. Moreover the levels of quality can be domain-dependent, because a level of quality can be good for some systems can be bad for others. Therefore, some levels can change while the application is running, for example the availability of server is very dynamic because it can manage request queues and the availability may depend on these queues.

In traditional software engineering many studies have been made to design quality metrics. Indeed, there exist models and methodologies that can be used to estimate the values of the quality attributes, such as the reliability of a system or the availability of a component. All these metrics are very useful to compare different systems, applications and patterns, but the most impor-

tant models are those that allow to provide clear relationships from design components to high level quality attributes [26], such as QMOOD (Quality Model for Object-Oriented Design) for Object-Oriented programming.

Unfortunately, these models are not applicable to SOA systems because of the inherent differences between the paradigms. Concepts such as class, methods, attribute or inheritance are commonly found in OO systems yet non-existent in SOA [26], so it is not possible to use the same metrics without modifications. Furthermore a significant difference is in the abstraction levels used to model system functionality; SOA was introduced to handle rapid requirements changes in the business, which is in contrast to Object-Oriented designs which aim to increase component reusability at much finer granularity. The SOA's dynamism leverages the overall system qualities because components of the architecture may change rapidly. Since the choice of a service can be made at runtime, the quality of a system can change. For these reasons the architect has to consider these aspects at design-time to satisfy the quality attributes required. It is not easy to estimate the quality of a Service-Oriented Architecture at design-time because some services of the system can be managed by third-parties and the risk that their level of quality may change is high [12]. In addition SOA metric models are applicable only after the system is implemented [26], since they use data which can only be gathered during dynamic analysis.

The interoperability is a quality attribute, therefore all the previous issues affect it. The level of interoperability in SOA cannot be guaranteed at design-time because designers do not always know all the services that will compose the architecture or they do not know how the system composition will change. Hence, the quality of the interoperability will depend on the components that in any time are present in the system. Its level can change a lot and continuously due to the heterogeneity of the elements and many problems can arise when at least one of these elements is managed by third-parties.

Chapter 5

Conclusions.

Analyzing the findings of the systematic literature review, we can assert that there is a strong impact of SOA on interoperability and on all of the differences involved in this assessment. The most relevant ones are those concerning the integration of services, independent elements that compose the software system. In particular, open world assumptions of SOA allow us to make important remarks about interoperability. As a matter of fact, we can consider services as building blocks of the system, without taking into account their implementations. In fact, in order to compose the final system, it is necessary to know what their functionalities are, not how they are executed. The use of standard interfaces enables strong decoupling between service provider and consumers and it defines the SOA principle of loosely coupling. To fulfill business goals, this tenet focuses design more on the integration of services than on their implementations. In addition, from loosely coupling derives another fundamental characteristic of SOA, dynamic binding. This allows services to interact with new ones also after deployment, at runtime. Service provider and service consumers may not share anything and in order to interact clients have to accept the contract exposed by the provider, which must ensure that its conditions are met during the consumption of the service. Moreover, the provider can offer more than one set of functional (or non-functional) requirements, in order to be consumed by many (potential) clients that are unknown at design-time. Furthermore, services may be managed by specialized third parties and executed remotely (since they are located within another organization), that is why the components of the system can be very heterogeneous. On the one hand, this heterogeneity can be exploited to improve flexibility in rapid business changes; on the other hand, it can cause many interoperability problems, which are different for each interoperability level.

The lowest one we considered in our study is the syntactic level. It is re-

lated to the data structure and message format employed to exchange information between entities. In our analysis we deduced that organizations make use of services for their interoperation capabilities. Interfaces and adapters play leading roles to guarantee syntactic interoperability because they translate custom service messages in formats accepted by data exchange protocols. Hence, even if the components are not related, syntactic interoperability is achieved and thanks to open standard protocols, such as XML and HTTP, it is easy to accomplish satisfactory solutions, within complex domains.

Semantic interoperability is harder to achieve. Despite service interfaces show which information can be exchanged, the client also has to understand the meaning of this information in order to decide whether to consume the service or not. Open world assumptions and runtime composition point out that the choice of services within a system is an issue to be addressed. We can see that semantic interoperability concerns mainly the phase of orchestration, rather than the data exchange between services. Most of the problems arise when a consumer has to choose which service to consume from many available providers because, in this case, there would be many conflicts in the representation of entities, properties and values. Indeed, the sender and the receiver do not know the subject domain model of the other one, since the information is described with ontologies that can assume different meanings within different contexts. Naming and identification conflicts occur frequently, even within the same domain. Despite this issue is heavily under research, there are no significant solutions for large-scale applications. These problems delay the adoption of Service-Oriented Architectures. In fact, nowadays most of the bindings are done at design-time, falling short of SOA expectations.

The last level we analyzed is pragmatic interoperability. Each level includes the underlying levels and, obviously, their problems, so previous issues affected pragmatic troubles. Besides data format and its meaning, pragmatic interoperability requires the interpretation of the information, which determines the transition to the next state of the system. This approach could model the system, for instance, as a finite state machine, a Petri net, etc. so that it is possible to define its behavior exploiting known techniques. The interoperability level would be better if a service could choose another one according to the consequences of this choice, rather than just its functionalities.

Thus, we can assert that difficulties for interoperability grow with its levels. We can affirm that syntactic interoperability in SOA has been achieved, whereas semantic and pragmatic have not completely been attained. However, despite pragmatic interoperability seems a very far goal to achieve, we can make several considerations about semantic. Many solutions for this

problem are currently being studied and some models and technologies are spreading also in real applications. This research field is related not only to Service-Oriented Architectures, but it relies on more generic issues concerning the representation of the ontology within a given knowledge domain. Efficient solutions for that problem can be useful to improve the quality of the service integration, obtaining systems that are more flexible, low-cost and faster both to be realized and changed.

Since Web services are the most widespread implementation of SOA, we want to make some remarks about their interoperability. Syntactic level is achieved through standard interfaces defined with WSDL, whereas interactions between services rely on SOAP. These XML-based standard protocols guarantee the communication between heterogeneous applications and specifications, like WS-I, improve this level significantly. Instead, for semantic interoperability there exist various implementations, based on OWL, that try to solve this problem. Unfortunately, none of them are able to provide good solutions when the environment is enlarged or the domain becomes more generic, essentially because they have to map all the ontologies of every service. Instead, pragmatic interoperability is not yet achieved, may be for the lack of its prerequisites in the current implementations, such as technical collaborations between the processes of different parties. Although Web services are claimed to be a good solution for integration of heterogeneous applications, we think that all previous considerations have to be further analyzed because, sometimes, the adoption of these technologies cannot cope with all interoperability issues.

Bibliography

[1] Uwe Zdun, Carsten Hentrich, and Wil van der Aalst. A survey of patterns for service-oriented architectures. *International Journal of Internet Protocol Technology*, pages 132–143, 2006.

[2] Edwin Morris, Linda Levine, Patrick Place, Daniel Plakosh, and Craig Meyers. System of systems interoperability (sosi): Final report. *TECHNICAL REPORT*, 2004.

[3] IEEE. The authoritative dictionary of ieee standards terms seventh edition. *IEEE 100*, 2000.

[4] Sujatha Kuppuraju, Aravind Kumar, and Geetha Presenna Kumari. Case study to verify the interoperability of a service oriented architecture stack. *IEEE International Conference on Services Computing (SCC 2007)*, 2007.

[5] Wenguang Wang, Andreas Tolk, and Weiping Wang. The levels of conceptual interoperability model: Applying systems engineering principles to m&s. *Spring Simulation Multiconference*, 2009.

[6] Sami Bhiri, Walid Gaaloul, Mohsen Rouached, and Manfred Hauswirth. Semantic web services for satisfying soa requirements. *International Federation for Information Processing*, pages 374–395, 2008.

[7] Leire Bastida, Alberto Berreteaga, and Inigo Cañadas. Adopting service oriented architectures made simple. *Enterprise Interoperability III*, pages 221–230, 2008.

[8] Jolita Ralyté, Per Backlund, Harald Kühn, and Manfred Jeusfeld. Method chunks for interoperability. *Springer*, pages 339–353, 2006.

[9] George Athanasopoulos, Aphrodite Tsalgatidou, and Michael Pantazoglou. Interoperability among heterogeneous services. *IEEE International Conference on Services Computing (SCC'06)*, 2006.

[10] Boualem Benatallah and Motahari Nezhad. Service oriented computing opportunities and challenges. *Springer*, pages 1–8, 2005.

[11] Hamid Motahari Nezhad, Boualem Benatallah, Fabio Casati, and Farouk Toumani. Web services interoperability specifications. *Computer (IEEE Computer Society)*, pages 24–32, 2006.

[12] Liam O'Brien, Paulo Merson, and Len Bass. Quality attributes for service-oriented architectures. *International Workshop on Systems Development in SOA Environments (SDSOA'07)*, 2007.

[13] Sriram Balasubramaniam, Grace Lewis, Ed Morris, Soumya Simanta, and Dennis Smith. Challenges for assuring quality of service in a service-oriented environment. *PESOS'09*, pages 103–106, 2009.

[14] Stanislav Pokraev, Dick Quartel, Maarten Steen, and Manfred Reichert. Requirements and method for assessment of service interoperability. *Springer*, pages 1–14, 2006.

[15] Meenakshi Nagarajan, Kunal Verma, Amit Sheth, John Miller, and Jon Lathem. Semantic interoperability of web services - challenges and experiences. *ICWS '06 Proceedings of the IEEE International Conference on Web Services*, 2006.

[16] Qing Gu and Patricia Lago. On service-oriented architectural concerns and viewpoints. *European Conference on Software Architecture. WICSA/ECSA 2009*, pages 289–292, 2009.

[17] Grace Lewis, Edwin Morris, Soumya Simanta, and Lutz Wrage. Common misconceptions about service-oriented architecture. *Sixth International IEEE Conference on Commercial-off-the-Shelf (COTS)-Based Software Systems (ICCBSS'07)*, 2007.

[18] Rodrigo Mantovaneli Pessoa, Eduardo Silva, Marten van Sinderen, Dick Quartel, and Luís Ferreira Pires. Enterprise interoperability with soa: a survey of service composition approaches. *Enterprise Distributed Object Computing Conference Workshops*, pages 238–251, 2008.

[19] Jameela Al-Jaroodi, Nader Mohamed, and Junaid Aziz. Service oriented middleware trends and challenges. *Seventh International Conference on Information Technology*, pages 974–979, 2010.

[20] Giovanni Denaro, Mauro Pezzè, and Davide Tosi. Ensuring interoperable service-oriented systems through engineered self-healing. *ESEC-FSE'09*, pages 253–262, 2009.

[21] Andreas Winter and Jürgen Eber. Using metamodels in service interoperability. *13th IEEE International Workshop on Software Technology and Engineering Practice (STEP'05)*, pages 1–10, 2005.

[22] Marijn Janssen and Hans Jochen Scholl. Interoperability for electronic governance. *ACM*, pages 45–48, 2007.

[23] Sven De Labey and Eric Steegmans. Making service-oriented java applications interoperable without compromising transparency. *Enterprise Interoperability III*, pages 233–245, 2008.

[24] Luciano Baresi, Elisabetta Di Nitto, and Carlo Ghezzi. Toward open-world software: Issues and challenges. *Computer (IEEE Computer Society)*, pages 36–43, 2006.

[25] Tom Goovaerts, Bart De Win, and Wouter Joosen. A flexible architecture for enforcing and composing policies in a service-oriented environment. *International Federation for Information Processing*, pages 253–266, 2007.

[26] Bingu Shim, Siho Choue, Suntae Kim, and Sooyong Park. A design quality model for service-oriented architecture. *15th Asia-Pacific Software Engineering Conference*, pages 403–410, 2008.

[27] Stanislav Pokraev, Dick Quartel, Maarten Steen, and Manfred Reichert. A method for formal verification of service interoperability. *IEEE International Conference on Web Services (ICWS'06)*, 2006.

[28] Liyi Zhang, Si Zhou, and Mingzhu Zhu. A semantic service-oriented architecture for enterprise application integration. *Second International Symposium on Electronic Commerce and Security*, pages 102–106, 2009.

[29] Balázs Simon, Zoltán László, Balázs Goldschmidt, Károly Kondorosi, and Péter Risztics. Evaluation of ws- standards based interoperability of soa products for the hungarian e-government infrastructure. *Fourth International Conference on Digital Society*, pages 118–123, 2010.

[30] Aphrodite Tsalgatidou and Eleni Koutrouli. Interoperability and eservices. *International Federation for Information Processing*, pages 50–55, 2005.

[31] Olaf Zimmermann, Vadim Doubrovski, Jonas Grundler, and Kerard Hogg. Service-oriented architecture and business process choreography in an order management scenario. *ACM*, pages 301–312, 2005.

[32] Jian Wang, Keqing He, Yangfan He, and Chong Wang. Towards service-oriented semantic interoperability based on connecting ontologies. *International Conference on Interoperability for Enterprise Software and Applications China*, pages 28–33, 2009.

Appendix A

Selected Sentences.

Here we can find all sentences selected from the systematic literature review.

BUILDING BLOCKS

Paper	Diff	Sentences
2	1	Services are the fundamental building blocks of software systems when applying the Service-Oriented Computing (SOC) paradigm
8	1	Given the architectural elements, or building blocks, any number of systems can be developed based on this architectural pattern
13	1	eServices are the building blocks for loosely-coupled, distributed applications based on the Service Oriented Architecture (SOA) principles
13	1	eServices are the building blocks of SOA and are mainly instantiated by Web Services (WS), Grid and P2P Services which are briefly described below
13	1	In this paper we investigated the interoperability potentials and challenges of WS, P2P and Grid services which are the building blocks of SOA and are known as eServices

continued on next page...

BUILDING BLOCKS

... continued from previous page

Paper	Diff	Sentences
19	1	The fundamental elements of this computing approach are loosely coupled software components, called services

Table A.1: Building Blocks.

COMPONENT MODELING

Paper	Diff	Sentences
1	1	For instance, concepts such as class, method, attribute, or inheritance are commonly found in OO systems yet non-existent in SOA systems
1	1	Another significant difference is in the abstraction levels used to model system functionality
8	4	Modifying the legacy system to replace internal functions with calls to external services, however, is much more technically challenging and likely more expensive
17	1,3	A service-oriented architecture (SOA) is an architectural approach for building systems where there are components that are service users and/or service providers
21	1	Service Oriented Architecture (SOA) is a promising model for enabling software vendors to present their software applications as services
21	3	Service Oriented Architecture (SOA) is not only an architecture, rather it is a relationship between the service provider, broker and user

continued on next page...

COMPONENT MODELING

... continued from previous page

Paper	Diff	Sentences
21	3	SOA basically involves three main players: the service provider, the service broker and the service requester
21	3	A relationship is created among participants: service provider, discovery agency, and the service requester

Table A.2: Component Modeling.

COMPOSITION

Paper	Diff	Sentences
4	1	Composition defines a composition of services into an executable workflow (business process)
5	1	Service Oriented Architecture (SOA) is an approach focused on software development to build loosely-coupled distributed applications using a collection of services
6	1	These products need to interoperate with each other for the system to behave as a cohesive whole and provide the desired functionality
6	1	An SOA implementation will have many services and these services have to interoperate in an SOA environment
8	1,2	If there are several providers of the same service the service consumer can choose at runtime which one to use
8	1,2	It is easy to compose services dynamically at runtime
8	1,2	In the case of dynamic binding, discovery and composition of services are done at run-time

continued on next page...

COMPOSITION

... continued from previous page

Paper	Diff	Sentences
8	2	More advanced automatic discovery and composition of new services at runtime requires the use of ontologies to describe function and usage of services
8	1	Composite business services may use one or more infrastructure services internally in addition to other business services
8	1	Building an application based on services can be like putting together a jigsaw puzzle where the parts do not quite fit
10	2	An issue that apparently is not being widely addressed is the support to end-users' service composition at runtime
10	1	We claim that service composition plays a major role in enterprise interoperability, and so here we present some state of the art on service composition approaches
10	1	In order to satisfy more demanding needs or to rapidly adapt to changing needs it is possible to perform service composition in order to combine the capabilities provided through several available services
10	1	Service composition is an essential ingredient of SOA, as it is concerned with aggregating interoperable services such that the goals of (enterprises in) a collaboration endeavour can be satisfied
10	1	A composite service consists of a composition of existing services to achieve some functionality that typically is not provided by a single available service
12	1	Each of these levels describes specific interoperability concerns which need to be tackled when integrating two service-oriented systems
12	1	Specifically, the first classification framework takes an internal look on the aspects that need to be considered when dealing with the integration of two systems, whereas the second one uses a system architect 'coarse' point of view for the identification of the various levels which are affected by the integration

continued on next page...

COMPOSITION

... continued from previous page

Paper	Diff	Sentences
14	1	All these organization comprise hundreds, thousands, or even more applications that need to communicate with each other
16	6	As a consequence, non-interoperable applications were created based on decentralised data management
18	1	The latter qualification becomes necessary because a composite system has properties that emerge due to the interaction of its components
19	1	It is an approach to building software systems that is concerned with loose coupling and dynamic binding between components (services) that have been described in a uniform way and that can be discovered and composed
19	2	The software then uses this information to dynamically access the service
19	2	Semantics bring closer the possibility of switching services dynamically by discovering them at runtime
19	3	Any collection of services needs common design, discovery, composition, and binding principles since they are typically not all developed at the same time
20	1	Companies A and B after discovering their match for business (e.g., using a public or private registry), need to agree on the joint business process, i.e., activities, message exchange sequence and interaction contracts, e.g., security, privacy and QoS policies
20	1	When services are described and interact in a standardized manner, the task of developing complex services by composing other (basic or composite) services is considerably simplified
21	1	The main advantage of this approach is giving the applications a way to integrate various services available online within the context of the applicationŠs specific domain and using them as needed instead of implementing the whole solution from scratch

continued on next page...

COMPOSITION

... continued from previous page

Paper	Diff	Sentences
21	1	The interactions among services are done through a standard description language such as XML, which makes it easy to integrate different services to build a business application and address problems related to the integration of heterogeneous applications in a distributed environment
24	1	The shift from developing large monolithic systems towards service oriented architectures, including the potential of web service-based implementations, leads to new chances and challenges in software development
25	1	Developers can use this combination of specifications to specify choreographies among multiple services

Table A.3: Composition.

DIFFERENT SERVICE CONSUMERS

Paper	Diff	Sentences
4	3,7	There are different groups of stakeholders which use the functionality of the architecture for various purposes
4	3	Different types of engineers could be involved in this process ranging from domain experts (ontology modeling, creation), system administrators (ontology deployment, monitoring) and software engineers

continued on next page...

DIFFERENT SERVICE CONSUMERS

... continued from previous page

Paper	Diff	Sentences
5	6,7	In exchange, these systems are characterized by being accessible by different and multiple users
6	7	Products support different versions of web service standards and specifications
8	7	What is the process for creating, evolving, and changing services if there are many consumers of the service?
9	7	Enabling interoperability with multiple implementations of a standard API shares some aspects with supporting evolving APIs, but entails additional challenges that call for novel approaches, as the one proposed in this paper
10	4,7	A service may have different implementations, and each implementation may have multiple deployments in different service end-points
10	7	The artefact produced in the requirements analysis phase is a software requirements document, which typically details the system's functional and non-functional requirements in a structured form
13	7	Grid services interoperability can be viewed along two different dimensions: between distributed resources in a Grid application, and between different Grid applications
19	1	Implementing a service-oriented architecture can involve developing applications that use services, making applications available as services so that other applications can use those services, or both
21	7	Services can implement a single business process or a set of different processes that are made available for integration with other heterogeneous services
21	7	This paper reviews the current work in the area and the trends and challenges to be addressed when designing and developing SOA middleware solutions for different application domains

continued on next page...

DIFFERENT SERVICE CONSUMERS

... continued from previous page

Paper	Diff	Sentences
25	7	A service's interoperability information has two types of consumers (and usage scenarios): human users and applications

Table A.4: Different Service Consumers.

DYNAMIC BUSINESS PROCESS

Paper	Diff	Sentences
1	2	Organizations face various business challenges in rapidly changing environments, often in forms of partnership changes or Merger & Acquisitions
1	2	Capable of providing mechanisms for integrating legacy systems at low cost or handing rapid business changes effectively, SOA mitigates risks introduced by fluctuations in the business world
2	2	Systems need to be able to comply with an ever growing set of business rules and regulations that are subject to continuous change
4	2	Changes to models and services are inevitable over time
5	2	Even though this kind of development requires an additional effort from a more traditional development, the resulting applications are flexible and capable of adapting in runtime to different needs, making them optimal and profitable

continued on next page...

DYNAMIC BUSINESS PROCESS

... continued from previous page

Paper	Diff	Sentences
7	2	Dynamic composition (and late binding in general) provides an exceptional challenge for verifying interoperability
7	2	The activities to verify syntactic, semantic, and organizational interoperability must be performed "on the fly" against the specific scenarios presented by the dynamic composition
9	2	Design for change approaches support the design of evolving APIs
9	2	Approaches to interchangeable services support interoperability of applications with services other than the ones the application was originally written for, but the work done so far considers mostly structural mapping between non-standard service interfaces, and dismisses semantic aspects of different implementations of standard interfaces
10	2	The binding phase may be performed either at design-time or at runtime, and can be static or dynamic
10	2	Dynamic service bindings allow a dynamic binding of service user and service provider's service at runtime, given a selection and discovery mechanism, usually defined at design-time
13	2	Furthermore, new services can be created and dynamically published and discovered without disrupting the existing environment
15	2,7	Second, WSIF determines the underlying protocol dynamically, so it can speak other protocols than SOAP, and it can react on protocol changes by switching between protocols at runtime
15	2	Currently, the programmer is forced to decide on the communication protocol at implementation time , leading to interoperability problems in architectures where services may decide to switch to other protocols at runtime

continued on next page...

DYNAMIC BUSINESS PROCESS

... continued from previous page

Paper	Diff	Sentences
17	2	Lifecycle management comprises design time (identifying and developing services), run time (defining and monitoring SLAs) and change time (upgrading and evolving services)
24	2	Web services form the most widespread infrastructure to enable components to collaborate dynamically
25	2	The general consensus is that this will change in the near future, with some dynamic binding possible as more services become available

Table A.5: Dynamic Business Process.

HETEROGENEOUS COMPONENTS IN THE SYSTEM

Paper	Diff	Sentences
2	7	A SOA interconnects a set of heterogeneous systems that may use different messages and formats
3	3,6	Semantic interoperability problems arise when the message sender and receiver have a different conceptualization or use a different representation of the entity types, properties and values from their subject domains
5	1	These Service based Systems, also called composite services, integrate and compose different existing services
6	6	Interoperability is the ability of software and hardware on various machines from various vendors to communicate with each other without significant changes to either side

continued on next page...

HETEROGENEOUS COMPONENTS IN THE SYSTEM

... continued from previous page

Paper	Diff	Sentences
6	6	Usually open Web Service standards and specifications are used to connect these products in an SOA stack However, interoperability cannot be guaranteed due to various reasons like differences in the versions of Web Service standards and specifications supported, differences in error handling mechanisms, differences in protocol support etc
6	7	If it does not interoperate, alternate solutions like adapter between products, product customization by the vendor or similar product from different vendors can be decided
8	1,3	Service implementations may involve developing new software, wrapping a legacy software system, incorporating services provided by third parties, or a combination of these options
8	5	Testing services based on heterogeneous technologies and owned by various organizations in an asynchronous and distributed environment is a non-trivial task
9	6	Service interchangeability focuses on interoperability between applications and services that fulfil equivalent goals, but are designed independently by different vendors and are not always fully compatible
12	6	Section 3 presents the interoperability dimensions that are involved when trying to integrate heterogeneous services
12	1	Let us consider for example the interoperability problem that may arise when integrating two systems that implement two incompatible processes
12	1	Thus, in order to facilitate the integration of heterogeneous services, special care should be given on these levels
12	1,3	Existing component-based technologies have tried to address this issue, but they haven't managed to provide a widespread solution that would enable the interoperation of diverse components developed by different providers, in multi-vendor platforms

continued on next page...

HETEROGENEOUS COMPONENTS IN THE SYSTEM

... continued from previous page

Paper	Diff	Sentences
13	1	Interoperability between different peers needs advanced interoperability techniques, since the various heterogeneous nodes of a P2P network need to communicate, exchange content and aggregate their diverse resources, such as computing power or storage space
13	6	One of the major benefits they offer is interoperability both between components of service oriented systems and between different systems
13	6	Thus, WS technology provides a means of interoperating between different software applications, running on a variety of platforms and/or frameworks
14	5	In the past, agencies could not exchange information because they operated widely disparate hardware that was incompatible
14	3,5	Interoperation occurs whenever independent or heterogeneous information systems or their components controlled by different jurisdictions/administrations or by external parties smoothly and effectively work together in a predefined and agreed upon fashion
15	3	This lack of transparency forces programmers to consider heterogeneity problems over and over again, even though interoperability is ideally a middleware responsibility
15	6	It is clear that such frameworks provide increased interoperability at the cost of decreased transparency
15	6	Object-oriented programming languages lack high-level support for platform-independent service interactions
16	6	However, true interoperability is not yet here since enterprises running different applications built with different designs and architectures still have difficulties talking to each other

continued on next page...

HETEROGENEOUS COMPONENTS IN THE SYSTEM

... continued from previous page

Paper	Diff	Sentences
17	1,4	In this context, a service is a distributed component with the following characteristics: is self-contained; has a published interface that abstracts the underlying logic; is location transparent; can be implemented in different languages or platforms and still interoperate; is discoverable and dynamically bound
17	5	However, the Web services goal of cross -vendor and cross-platform interoperability begins to fall short when services start to use features beyond the two basic standards: Web Service Definition Language (WSDL) and Simple Object Access Protocol (SOAP)
18	1	Interoperability is the capability of multiple, autonomous and heterogeneous systems to use each other's services effectively
19	6	In order to enable dynamic and seamless cooperation between different systems and organizations, implementing SOA poses new challenges to overcome
19	6	On the other hand, there is no universally agreed standard middleware, which makes it difficult to construct applications from components that are built using different programming models (such as Microsoft COM, OMG CORBA, or Java 2 Platform, Enterprise Edition (J2EE) Enterprise Java Beans)
19	6	Thus, a consumer can use the functionality offered by multiple providers without worrying about the underlying differences in hardware, operating systems, programming languages, etc
21	6	Abstractions to hide the heterogeneity of underlying environments thru supportive languages and protocols
24	6	The approach on using meta modeling technologies to enable service interoperability on data level is independent from the technological space used for service interaction

continued on next page...

HETEROGENEOUS COMPONENTS IN THE SYSTEM

... continued from previous page

Paper	Diff	Sentences

Table A.6: Heterogeneous Components in the System.

INTERFACES AND ADAPTERS

Paper	Diff	Sentences
2	6	Interoperability is achieved by inserting the adapter components (Message Adapters and Policy Service Adapters) that translate back and forth between the generic format and native formats
4	3	On the other hand, the group of engineers forms those stakeholders which perform development and administrative tasks in the architecture via Developer Interface
4	6	Different enterprise systems are connected through one interface, and a cross-system data transfer and the reusage of objects or components is enabled
5	2	This enables in SOA based systems to easily replace and add new services and changes without impacting on the service interface
13	1	The Open Grid Services Architecture (OGSA) is a significant effort by the Global Grid Forum towards the standardization of protocols and interfaces which integrates Grid and WSs
15	6	In this paper, we show that interoperability in Java applications can be achieved without compromising transparency

continued on next page...

INTERFACES AND ADAPTERS

... continued from previous page

Paper	Diff	Sentences
17	6	That is possible because Web services define the interface format and communication protocols but do not restrict the implementation language or platform
19	1,6	Services are software components with well-defined interfaces that are implementation-independent
19	6	A service is accessed by means of a service interface, where the interface comprises the specifics of how to access the underlying capabilities
19	6	A service is opaque in that its implementation is typically hidden from the service consumer except for (1) the information and behavior models exposed through the service interface and (2) the information required by service consumers to determine whether a given service is appropriate for their needs
21	1	It provides a framework to represent business processes as independent modules (services) with clear and accessible interfaces
22	6	However, in the past it was difficult to define component interfaces using self-describing, openly standardized interface specifications which are now available, for instance Web Services Description Language (WSDL) and Business Process Execution Language (BPEL) (to describe the workflow itself)
25	6	Management specifications provide for the definition of visible interfaces for service tracking, accounting, auditing, supervision, and control of service
25	6	Aside from such languages, these specifications donŠt make assumptions or impose constraints on how to define interfaces

Table A.7: Interfaces and Adapters.

MANY ORGANIZATIONS IN THE SYSTEM

Paper	Diff	Sentences
2	3,5	The applications that emerge in service oriented architectures can become large and fairly complex, and can be interconnected with services from various organizations and stakeholders
4	1,4,5	Depending on particular architecture deployment and integration scenarios, the back-end applications could originate from one organization (one service provider) or multiple organizations (more service providers) interconnected over the network (internet, intranet or extranet)
4	5	The architecture thus can serve various requirements not limited to Enterprise Application Integration, but also Business to Business (B2B) integration
5	1,5	Moreover, the Service based Systems are composed by several applications and distributed services from different organisations; therefore the control and management of the systems is a critical issue
7	5	Achieving organizational interoperability further requires that the business activities that cross applications and services not only be aligned across organizations but also well understood by the testers
8	5	The goal of these standards is to ensure that Web Services, tools, and runtime environments interoperate across vendors and organizations
8	4,5	Typically, services are reused across applications that often cross enterprise boundaries
10	5	SOA is based on the assumption that enterprise systems may be under the control of different ownership domains

continued on next page...

MANY ORGANIZATIONS IN THE SYSTEM

... continued from previous page

Paper	Diff	Sentences
10	5	Different organisations define ontologies in different ways, which may generate major problems of interoperability
11	5	The proposed architecture of the Hungarian e-Government Framework, mandating the functional cooperation of independent organizations, puts special emphasis on interoperability
11	5	Although Hungary has middle-ranked position in the level of e-government services, strategic studies and assessments showed that one of the primary deficiencies is the lack of interoperable, multi- and cross-organizational back-office functionality
11	5	The process-level layer orchestrates cross-organizational activities and services
14	5	Departments and institutions collaborate and interoperate in processes crossing their organizational boundaries
16	5	Not only large organisations set up cooperation agreements with other enterprises, also small and medium sized enterprises are combining their forces to compete jointly in the market
16	5	Interoperability is an issue that arises when multiple organisations need to cooperate via information systems
19	5	Especially, in the case of service interaction where the message and information exchanges are across boundaries, a critical issue is the interpretation of the data
19	5	Its strong decoupling between service provision and consumption enables much more flexible and cost-effective integration, within and across organizational boundaries, than existing middleware or workflow systems do

continued on next page...

MANY ORGANIZATIONS IN THE SYSTEM

... continued from previous page

Paper	Diff	Sentences
20	1,5	Recent advances in Web service technologies provide necessary building blocks for supporting the development of integrated applications within and across organizations

Table A.8: Many Organizations in the System.

OPEN WORLD

Paper	Diff	Sentences
2	5	Moreover, the underlying resources that are being interconnected may be implemented on different heterogeneous systems that are unaware of each other
3	3,6	However, the message sender does not always know the subject domain model of the message receiver
3	6	Depending on its knowledge, the message sender makes assumptions about the subject domain model of the receiver and uses them to construct a message and to communicate it
5	6	Service deployment: This task makes the service available in a suitable runtime environment in order to receive incoming requests from potential consumers
8	6	Changes in service interface and implementation must be tested continuously by each of the service consumers in order to ensure that the actual service behavior conforms to intended behavior

continued on next page...

OPEN WORLD

... continued from previous page

Paper	Diff	Sentences
12	4,6	Intended Clients: Although there might be specific security constraints dictating a different case, web services in general may be invoked by any client with internet access, provided that a client has the necessary infrastructure (e.g. a SOAP engine) to exchange messages (e.g. SOAP messages) with the service provider
12	6	Context level interoperability is important when dealing with service interoperability in ubiquitous computing
13	6	Service Oriented Architectures (SOA) emerged as an evolutionary step from Object and Component based approaches, with the promise to support the loose coupling of system parts and to provide agility, flexibility and cost savings via reusability, interoperability and efficiency
15	6	In this paper, we investigate how the lack of transparency in object-oriented programming languages can be cured, taking Java as an example
18	6	However, the message sender does not always know the subject domain model of the message receiver
18	6	Depending on its knowledge, the message sender makes assumptions about the subject domain model of the receiver and uses this assumed subject domain model to construct a message and to communicate it
19	6	Unlike objects or databases, a service is developed for use by its consumer, which may not be known at the time
19	6	Loosely coupling means that services interactions are neither hard coded (like in Object Oriented Programming), nor specified at design time (like in Component Based Modelling)

continued on next page...

OPEN WORLD

... continued from previous page

Paper	Diff	Sentences
19	6	Services are loosely coupled: Services are designed to interact without the need for tight, cross-service dependencies
19	6	What distinguishes SOA from other architecture paradigms is loose coupling
19	6	Loose coupling means that the client of a service is essentially independent of the service
19	6	The difference between the Web services approach and traditional approaches (for example, distributed object technologies such as the Object Management Group - Common Object Request Broker Architecture (OMG CORBA), or Microsoft Distributed Component Object Model (DCOM)) lies in the loose coupling aspects of the architecture
19	6	A service is provided by an entity - the service provider - for use by others, but the eventual consumers of the service may not be known to the service provider and may demonstrate uses of the service beyond the scope originally conceived by the provider
23	6	Note that the relations between the sub-ontologies Oi and the RSO are loosely coupled
25	3	For example, the SOAP middleware and its prebuilt libraries support the sending and receiving of messages, so developers don't need to know these details to implement Web services
25	6	In general, a client needs all interoperability specifications at binding time because it needs to know whether a service supports a certain specification

Table A.9: Open World.

POLICIES

Paper	Diff	Sentences
2	6,7	Due to the openness and very frequent evolution of a service oriented environment, it needs to be able to interface with a range of policy languages, policy servers, message formats and functional services
2	7	Therefore, a policy enforcement solution needs to be interoperable with multiple message formats and with multiple policy services
2	6,7	The policy enforcement service needs to bridge a variety of message formats on the one hand, and different types of requests for policy services on the other hand
25	7	A service definition can include policies-for example, privacy policies-and other nonfunctional properties-for example, QoS descriptions such as response time-that clients interacting with a service should understand

Table A.10: Policies.

SERVICES ARE LOCATED DIFFERENTLY

Paper	Diff	Sentences
3	4	In general, the interaction mechanism is identified by its location (e.g. the combination of an IP address and port number can be used to identify a TCP/UDP socket)
4	4	Generally, the goal is to allow users to interact with business processes on-line while at the same time reduce their physical interactions with back-office operations

continued on next page...

SERVICES ARE LOCATED DIFFERENTLY

... continued from previous page

Paper	Diff	Sentences
5	4,5	In some cases the company developing the service will host the service itself or may choose to work with a third party as a service provider
8	3	These services then can either be developed in-house or bought from external service providers
10	4	The execution phase involves the invocation of all participating services, possibly hosted in different provider domains
19	4	A service-oriented architecture represents an abstract architectural concept defining an information technology approach or strategy in which applications make use of (perhaps more accurately, rely on) services available in a network such as the World Wide Web

Table A.11: Services Are Located Differently.

SYSTEM QUALITY

Paper	Diff	Sentences
1	1,2	A SOA system's quality depends on the appropriateness of its design, which should be evaluated and managed from early in its development
1	1,2	However, finding the right SOA-based metric model that quantifies overall system qualities and is applicable early in development is difficult
1	1,2,4	Failure to apply OO metric models to SOA systems led to the recent introductions of several SOA-based metrics, but most of these emergent researches are applicable only after the system is implemented

continued on next page...

SYSTEM QUALITY

... continued from previous page

Paper	Diff	Sentences
4	2	Services can be chosen at run time based upon their availability or other factors, such as quality
8	3	It is the architect's responsibility to understand the quality attribute requirements and architect the system around the tradeoffs that are most important to the stakeholders of the system
17	3	Building a system that relies on third parties without the necessary agreements increases the risk of not meeting the quality attribute requirements

Table A.12: System Quality.

Appendix B

Pilot study.

Introduction.

After analyzing the differences between traditional software engineering and service-oriented system engineering we decided to study their impact on a particular software quality attribute as the interoperability. The procedure adopted was to do initially a systematic literature review choosing most relevant papers about SOA and interoperability. Afterwards we filtered suitable papers and we selected sentences and considerations that proved differences between TSE and SOSE and especially that highlighted the impact of these differences on the interoperability. Then we made a table to link every single sentence with the difference(s) that involves the attribute. Actually, sentences have been grouped according by a common topic (with a title and a corresponding letter). In the table the letter alone means that all sentences about that topic are linked whereas a letter followed by a number refer to a single sentence. Finally we wrote our considerations about the choice of every link focusing the differences regarding traditional software and service oriented components. Moreover we described the impact that these differences have on the interoperability.

Analysis.

a) INTERFCES DESCRIPTION AND ADAPTERS

1. The basic concept of a service-oriented architecture (SOA) is quite trivial: a service is offered using a remote interface that employs some

	Diff A	Diff B	Diff C	Diff D	Diff E	Diff F	Diff G
Interoperability	b	a5	c	a1	f	n	a1
	c2	d7	d5	c1	g1		a2
	d	o3	g4	g2	g3		a3
	g5			n2	h		a4
	n1			n4	g8		a6
	o1						e
							g6
							i
							l1
							m
							o2

Table B.1: Pilot Table.

kind of well-defined interface description. [1]

2. Communication channels and messages are usually described with interface descriptions. The interface description of a SOA needs to be more sophisticated than the interface descriptions of (OO-)RPC distributed object middleware, however, because it needs to be able to describe a wide variety of message types, formats, encodings, payload, communication protocols, etc. [1]

3. Both client and server applications may have to support many different service adapters and service interfaces, supporting different models. [1]

4. A characteristic property of SOAs is that they are highly adaptable in the remoting layer. Possibly different communication protocols and styles must be supported, even at the same time. [1]

5. A SOA usually has to be able to be adapted at runtime. [1]

6. Variation at the communication layer is usually handled via protocol plug-ins. Protocol plug-ins extend the client request handler and server request handler with support for multiple, exchangeable communication protocols. [1]

b) BACKENDS

1. A service provider offers a service to service clients. Often the service is not realized fully by the service provider implementation, but also by a number of backends, such as server applications (other SOAs or

middleware-based systems such as CORBA or RMI systems), ERP systems, databases, legacy systems, and so forth. Flexible integration of heterogeneous backend systems is a central goal of a SOA. Even though the use of backend systems is of course optional, it is an important characteristic of SOAs. [1]

2. The service provider is the remote object realizing the service. Often the service provider does not realize the service functionality solely, but instead uses one or more backends. When a SOA is used for integration tasks, it should support multiple backend types. [1]

c) BROKER ROLE

1. Remoting. This layer implements the middleware functionalities of a SOA (for instance a Web services framework). Usually, these details of the client side and the server side are hidden in a broker architecture: a broker hides and mediates all communication between the objects or components of a system. [1]

2. Concerning integration of SOA and business processes there are several important integration patterns, such as router, broker, and managed process. These are general patterns that are, in combination, suitable for bridging the two views of SOA and business processes. [1]

3. Concerning the microflow level, the broker and router patterns are important in order to model communication between a process-step and services at an endpoint at a technical level. The request for service invocation sent by the process-step must be routed to the right endpoint, which is done by a broker. [1]

4. Within this architectural pattern, various components connect to a service bus via their service interfaces. In order to connect those components to the bus, service adapters are necessary. The service bus handles service requests and generally represents a message-based router and/or broker. [1]

d) COMPOSITION

1. In the enterprise scope, often multiple SOAs and other (distributed) systems need to be composed to work together. [1] The backend does not need to be a legacy system or another non-SOA participant: the backend can be another service as well. This way, service composition can be realized architecturally using a distributed variant of the pattern component wrapper. [1]

2. Services come in two flavors: simple and composite services. The unit of reuse with services is functionality that is in place and readily available and deployable as services that are capable of being managed to achieve the required level of service quality. [6]In formal academic literature, a service is defined as a business function implemented in software, wrapped with a formal documented interface that is well known and known where to be found not only by agents who designed the service but also by agents who do not know about how the service has been designed and yet want to access and use it. These services could be simple services performing basic granular functions such as order tracking or composite services that assemble simple or other composite services to accomplish a modular business task such as a specialized product billing application. [4] As an example, a business flow, such as an online book retail service, could be built using services across multiple service providers pulling together, say, billing services from a partner, and warehousing services from another partner. [4]

3. Specifically, the roadmap details the following four areas - (i) Service foundations - the fundamental infrastructure that implements the connectivity of heterogeneous components; (ii) Service Composition - the aggregation of services into a single composite service, addressing control and data flow, and transaction integrity; (iii) Service Management and Monitoring - the myriad of activities required to control and monitor SOA applications and infrastructures from troubleshooting to auditing, and includes systems engineering attributes like scalability, performance, and availability; and (iv) Service Design and Development - tying the design and development of services to the business process, a key facet that can fully realize the benefits of adopting SOA as a business strategy. [4] A third study suggests the need for a collaboration and coordination fabric, but define it to be "a conceptual artifact that is used to connect interrelated entities by providing communication, coordination, and collaboration mechanisms". [4]

4. While the services encapsulate the business functionality, some form of inter-service infrastructure is required to facilitate service interactions and communication. [6]

5. If designers perform discovery at design time, they select the services and hard-code their addresses into the BPEL workflow. If they perform discovery at deployment time, they use a service broker to "configure" the application. [7]

6. Services can encapsulate component behavior at many levels, but still describe it in the same way, thus easing composition of the components. [12]

7. A new problem arises when we want to introduce new applications and configure them to interoperate. [12]

e) FRONTENDS

1. Sometimes a number of different frontends need to access one service. One special variant of multiple frontends is that there is more than one service offered, and each of the frontends is a different channel. [1]

f) DOMAINS

1. Organizations can benefit from Grid Computing and Cloud Computing in different domains: internal business processes, collaboration with business partners and for customer-faced services. [2]

2. However, this sort of calculation only makes sense if placed in a broader context. Whether or not computing services can be performed locally depends on the underlying business objective. It might for example be necessary to process data in a distributed environment in order to enable online collaboration. [2]

g) STANDARDS

1. When it comes to wider use across organizational boundaries, however, the use of these models are hampered by the lack of uniform standards and support from major software vendors. [4]

2. The World-wide Web Consortium formally defines a Web service as "a software system designed to support interoperable machine-to-machine interaction over a network". The interface to a Web service is described in a machine-processable format, specifically Web Services Definition Language or WSDL. Other systems interact with the Web service using SOAP messages (typically XML over HTTP) in conjunction with other Web-related standards. Web services typically have the following characteristics:

    - They are independent of the underlying transport protocol
    - The service attributes (location, capabilities, and access mechanism) are described in the XML-based WSDL

- Web services use the directory services standard Universal Description, Discovery, and Integration (UDDI) to facilitate discovery and use by clients, and
- They use XML over HTTP (SOAP) to communicate with each other

Although Web services have been characterized as old technology in a new implementation (distributed RPCs) or even broadly as middleware, Web services are essentially the deployment of a service-based computing model over the Internet, and unlike other earlier technology implementations leverage open Internet standards to facilitate diverse inter-enterprise communication and has garnered relatively unanimous industry vendor support. [4]

3. The efforts are too numerous to list here, but the various online trade journals have an abundance of information on vendor products, and ongoing collaboration efforts across various vendors to promote standards and interoperability for enterprise service infrastructures. [4]

4. For example, the elements in the SOA pattern include service consumers, service descriptions, service implementations, and possibly a service bus. One relationship is that between service providers and service consumers. In the case of Web Services, consumers and services are connected by HTTP or HTTPS connectors carrying SOAP messages. Given the architectural elements, or building blocks, any number of systems can be developed based on this architectural pattern. These concrete elements and their interactions are the architecture of the system. [5] Software architects still need to architect systems based on the SOA architectural pattern. They have to design services and service interactions that meet the qualities that stakeholders expect of the system. In addition, the architect(s) must make decisions on how services are implemented. Service implementations may involve developing new software, wrapping a legacy software system, incorporating services provided by third parties, or a combination of these options. Is it technically feasible to create a service from the legacy system or part of the system? How much would it cost to expose services from the legacy system? However, being stable for years does not mean that the standards are complete. For example, after adopting basic infrastructure Web service standards, some organizations found that their services still could not communicate information effectively with other services due to different design decisions and flexibility in the standards.

The WS-I Basic Profile was constructed to provide better interoperability across implementations using basic infrastructure standards. [5]

5. Once all the elements of an enterprise architecture are in place, existing and future applications can access these services as necessary without the need of convoluted point-to-point solutions based on inscrutable proprietary protocols. This architectural approach is particularly applicable when multiple applications running on varied technologies and platforms need to communicate with each other. In this way, enterprises can mix and match services to perform business transactions with minimal programming effort. [6] The ability to layer solutions and support heterogeneity allows for gradual migration to service-based solutions. The development of XML-based languages for defining and enforcing service-level agreements, workflow, and service composition is supporting the gradual change of business processes, envisioned as part of the growth of Software as a Service. [9]

6. Standards such as SOAP for Web services help to ensure that heterogeneity of solutions poses no problems. [9]

7. However, the Web services goal of cross-vendor and cross-platform interoperability begins to fall short when services start to use features beyond the two basic standards: Web Service Definition Language (WSDL) and Simple Object Access Protocol (SOAP). Over the last few years, many Web services standards have emerged from a number of standards bodies. [11]

h) ORGANIZATIONS

1. It follows that the adoption of service oriented computing cannot be without impact on the organization - and it appears that the impact could possibly be felt across the organization's internal boundaries and beyond them to the interfaces with partner organizations. [4]

2. Services can be reused across applications that cross enterprise boundaries. Changes requested by one service consumer in an existing service can result in undesired results for another service consumer. Changes in service interface and implementation must be tested continuously by each of the service consumers in order to ensure that the actual service behavior conforms to intended behavior. [5]

3. Since services maybe offered by different enterprises and communicate over the Internet, they provide a distributed computing infrastructure

for both intra and cross-enterprise application integration and collaboration. Clients of services can be other solutions or applications within an enterprise or clients outside the enterprise, whether these are external applications, processes or customers/users. [6]

4. Internally, the opportunity exists to increase organizational information systems' flexibility and adaptability. Externally, the opportunity exists to generate revenue from existing software and to flexibly and rapidly obtain new software without the burden of ownership. [9]

i) SEMANTIC

1. True interoperability can only be achieved if service consumers and providers interoperate at both the syntactic and semantic levels. [5]

2. Enriching the service interfaces with additional semantic information such as scenarios or behaviors, allows a more robust and stable service composition. [8]

3. Interoperability refers to the ability of a collection of communicating entities to share specific information and operate on it according to an agreed-upon operational semantics. [11]

4. Services offer programming abstractions in which software developers can create different software modules through interfaces with clearer semantics. [12]

l) MULTI-USER

1. Because services provide a uniform and ubiquitous information distributor for wide range of computing devices (such as handheld computers, PDAs, cellular telephones, or appliances) and software platforms (e.g. UNIX or Windows), they constitute the next major step in distributed computing. [6]

m) INTERFACE DEFINITIONS

1. Service-based applications are developed as independent sets of interacting services offering well-defined interfaces to their potential users. [6]

2. As service interfaces of composed services are provided by other (possibly singular) services, the service specification serves as a means to define how a composite service interface can be related to the interfaces of the imported services and how it can be implemented out of imported service interfaces. [6]

3. As service development requires that we deal with multiple imported service interfaces it is useful to introduce this stage the concept of a service usage interface. A service usage interface is simply the interface that the service exposes to its clients. [6]

4. In house service design and implementation. Once a service is specified, the design of its interfaces or sets of interfaces and the coding of its actual implementation happens in-house. [6]

5. Outsourcing service design and implementation. Once a service is specified, the design of its interfaces or sets of interfaces and the coding of its actual implementation may be outsourced. [6]

6. Grid services are stateful services that provide a set of well-defined interfaces and follow specific conventions to facilitate coordinating and managing collections of web-service providers/aggregators. The grid service indicates how a client can interact with it and is defined in WSDL. [6]

n) OPEN WORLD

1. Component-based software represents another major advance, moving software development processes to the open world. Third parties develop and provide components and are responsible for their quality and evolution. Application development thus becomes partly decentralized. At an extreme, application development consists of gluing components together by using middleware technology as the integration and coordination infrastructure. [7]

2. Researchers and practitioners are also increasingly interested in building applications by assembling existing services executed remotely at the provider site.

    - Developers and users must trust the services they use to compose the application. Each service should clearly describe its nonfunctional characteristics, as well as its functionality, to let the client understand if the service fits its needs. Moreover, the client must be assured that the service meets its description's promises.
    - Developers should define suitable mechanisms to set up and negotiate service-level agreements between clients and services.
    - Developers should allow applications to set the bindings to specific services at runtime.

- Because services might change unexpectedly and because of dynamic binding, their users (either humans or other applications/-services) need to monitor real behaviors that might deviate from what's expected and plan for strategies to react to them. [7]

3. The following are some existing standards, industrial products, and research prototypes that support, to a certain extent, the open world assumptions. Service-oriented technologies:

   - Jini
   - Open Services Gateway Initiative (OSGI)
   - SOAP
   - Web Service Description Language (WSDL)
   - Universal description, discovery, and integration (UDDI)
   - Business Process Execution Language (BPEL)
   - Web Service Security (WS-Security)
   - Web Services Trust Language (WS-Trust)[7]

4. Web services let designers integrate and remotely use services that different providers supply. The Web services can also be composed together to form more complex services. Designers typically use the Business Process Execution Language for this. BPEL imposes a workflow-based coordination of involved services and requires the identification of at least the structure of the WSDL interface of all parties at design time. In principle, designers can discover new services at different times. [7]

5. Heterogeneity. Any SSE concept, method or tool has to embrace heterogeneity of the service application and the context in which it operates. Just like dynamism, heterogeneity impacts all phases of the service development lifecycle, posing restrictions on how software service systems can be designed, developed, deployed, and evolved over time. Note that in contrast to current practice, no assumptions can be made about the system's programming, execution, and management context before, during or after deployment. [8]

6. In our view, SSE will be based on standards and will be frequently realized with Web services. In fact, languages such as SOAP, WSDL, BPEL, WS-Policy, WS-Agreement already constitute the first step to realize the technical aspects in some of the SSE tenets. However, it

is evident that research is needed to more effectively satisfy the open-world assumption. [8]

o) TENETS

1. Without sound SSE tenets, we cannot guarantee that the usage of SOA-compliant methods and tools results in software applications that meet the basic SOA criteria ensuring that services are loosely coupled, self-contained, and have a clean interface that is geared towards (re-) composition. [8]

2. Firstly, dynamism implies that SSE methods, techniques, and tools have to deal with emergent properties and behavior of complex service networks, which may in fact be comprised of thousands of independent - yet cooperating - services. Late binding and loose coupling constitute two key principles for increasing the adaptability of service applications, accommodating dynamic (re-) composition and (re-) configuration of services in a network. [8]

3. In addition, SSE has to deal with services that may be deployed on various run-time platforms, including mobile devices, computing clouds, and legacy systems, and have been developed in various programming paradigms - including, but not limited to, OO and CBD. [8]

4. Dynamism. A key tenet of SSE is dynamism regarding both the services that are aggregated into dynamic service compositions. [8]

Pilot study references.

[1] Zdun, U.; Hentrich, C. & van der Aalst, W. M. P., A survey of patterns for Service-Oriented Architectures, International Journal of Internet Protocol Technology, 2006, 1, 132-143

[2] Klems, M.; Nimis, J. & Tai, S., Do Clouds Compute? A Framework for Estimating the Value of Cloud Computing, 7th Workshop on E-Business (WeB2008), Springer, 2009

[3] Gu, Q. & Lago P., Exploring service-oriented system engineering challenges: a systematic literature review,SOCA (2009) 3:171Ű188, Springer, 2009

[4] Luthria, H. & Rabhi, F., Service Oriented Computing in Practice - An Agenda for Research into the Factors Influencing the Organizational Adoption of Service Oriented Architectures, Journal of Theoretical and Applied Electronic Commerce Research, 2009, 4, 39-56

[5] Lewis, G. A.; Morris, E.; Simanta, S. & Wrage, L., Common Misconceptions about Service-Oriented Architecture, Proc. Sixth International IEEE Conference on Commercial-off-the-Shelf (COTS)-Based Software Systems ICCBSS '07, 2007, 123-130

[6] Papazoglou, M. P., Service-oriented computing: concepts, characteristics and directions, Proc. Fourth International Conference on Web Information Systems Engineering WISE 2003, 2003, 3-12

[7] Baresi, L.; Nitto, E. D. & Ghezzi, C., Toward Open-World Software: Issue and Challenges, Computer, IEEE Computer Society, 2006, 39, 36-43

[8] van den Heuvel, W.-J.; Zimmermann, O.; Leymann, F.; Lago, P.; Schieferdecker, I.; Zdun, U. & Avgeriou, P., Software Service Engineering: Tenets and Challenges, ICSE International Workshop on Principles of Engineering Service Oriented Systems (PESOS), IEEE Computer Society, 2009

[9] Gold, N.; Mohan, A.; Knight, C. & Munro, M., Understanding service-oriented software, IEEE Software, 2004, 21, 71-77

[10] Gu, Q. & Lago P., On Service-Oriented Architectural Concerns and Viewpoints, European Conference on Software Architecture, WICSA/ECSA, 2009

[11] O'Brien Lero, L.; Merson, P. & Bass, L., Quality Attributes for Service-Oriented Architectures, Proc. International Workshop on Systems Development in SOA Environments SDSOA '07: ICSE Workshops 2007, 2007, 3

[12] Huhns, M. N. & Singh, M. P., Service-oriented computing: key concepts and principles, IEEE Internet Computing, 2005, 9, 75-81