Service contract clauses as business rules

Fabio De Rosa¹, Ariele P. Maesano², Libero Maesano³

¹ fabio.de-rosa@simple-eng.com - SIMPLE ENGINEERING, Rome, Italy
² ariele.maesano@lip6.fr - Laboratoire d’Informatique de Paris 6 UPMC, Paris, France
³ libero.maesano@simple-eng.com - SIMPLE ENGINEERING, Paris, France

Abstract: Service contracts between digital providers and consumers should be: (i) understood and verified by the business stakeholders, (ii) mechanically checked for consistency and (iii) automatically mapped to system implementations. This paper presents guidelines and templates for expressing service contract clauses as business rules, i.e. logical formulations in constrained natural language conforming to the OMG Semantics of Business Vocabulary and Business Rules (SBVR) specification. When supported by the SBVR abstract notation, natural language business rules can be mechanically checked for consistency, and eventually processed for automated generation of technical model elements, machine-readable components, executable code and tests. The service contract clauses are established through a method that is inspired by the Design by contract approach: the signature is expressed by means of a set of business vocabulary entries and the pre/post-conditions are stated as operative business rules. The method and the corresponding rule templates are included in a methodological framework for model driven engineering of services architectures (simpleSOAD®), in which the service contract, which is a bundle of rights and duties for service providers and consumers, is represented as a layered collection of formal models. These models work as a body of requirements for implementations. In this model-driven framework, the service contract vocabulary and rulebook constitute the service Computation Independent Model (CIM). We introduce the method guidelines and templates through an example and discuss the assumptions on the resulting conceptual model as well as some recommendations on the contract building enterprise. Moreover, we evoke the mappings between the service contract vocabulary and rulebook and (i) the service Platform Independent Model (PIM) - an abstract computational model expressed in UML/OCL notation - and (ii) the service interoperability Platform Specific Model (PSM) - an implementation model dependent on the underlying interoperability platform such as SOAP, REST, ... - as well as issues about requirement-based and model-based functional testing.

Keywords: business rules, CIM, contract-first, contract-based design, Design by contract, MDA, MDE, model-driven, OCL, PIM, PSM, SBVR, SOA, SoaML, testing, UML.

1. INTRODUCTION

In the digital economy, a growing number of applications, systems and devices are connected and collaborate without human intermediation, allowing the automation of business processes that support daily activities [5]. These digital actors have no mutual knowledge of their internals - they shall only agree on the services that they deliver each other. Service Oriented Architecture (SOA) is a design and implementation style that allows organizations to put in practice dynamic collaborations of autonomous, heterogeneous and loosely coupled digital actors in order to achieve flexible, dependable and secure business process automation.

Services can be roughly defined as activities performed by digital actors - acting as service providers - whose results can be used by other digital actors - acting as service consumers. These activities, including service invocation, deliberation (i.e. decision about service delivery), delivery/use and reporting, should be described independently from the internals of the digital actors that perform them. The service description concerns: (i) the operations whose results are delivered by the provider and used by the consumer, (ii) the provider and consumer external behaviors that invoke, undertake, decline, report, ... these operations and (iii) the interfaces where these behaviors are visible.

In order to build effective services architectures, the service descriptions should be deemed by the digital actors as contracts, i.e. bundles of rights and duties for the service parties. Service contracts should be practicable, i.e. able to be effectively implemented by the service parties, and enforceable, i.e. able to be confronted for compliance with the service parties’ actual interfaces and behaviors. Hence, service contract clauses should be taken into account as functional and non-functional requirements for digital actor design, implementation and testing - this is what is referred to as the contract-first approach.

The contract-first approach is mandatory when the digital actors are managed by independent organizations and their implementations are hidden from each other. It poses three related but conflicting challenges to service and system engineering. Firstly, the service contracts clauses - in particular the functional clauses that are the focus of this paper - should be formulated at the business level and in a form that should be easily understood and validated by the business stakeholders. Secondly, these clauses should be expressed in a formal way in order to
be mechanically checked for consistency and other formal properties. Thirdly, these clauses should be mapped to system implementation decisions in order to ensure their traceability as requirements for design and eventually to allow the automatic generation of machine readable elements, executable code and, last but not least, test cases and oracles.

The research and practice on these issues spans three originally distinct but now increasingly related research fields: (i) contract-based service design and implementation; (ii) business rules formulation, implementation and management; (iii) model-driven architecture and engineering.

The SOA research domain is broad and far from homogeneity. A large part of the involved researchers consider the service orientation as an evolution of the distributed architecture research field, i.e. an implementation issue - how to dynamically distribute and combine the computations in the most efficient ways. In this approach, services are massively distributed software components that can be dynamically combined, even in unexpected (emergent) manners.

The contract-first approach adopts a more realistic point of view: the problem is how to put in place the most flexible, dependable and secure cooperation between digital actors whose implementations can be preexistent - legacy systems - and are in any case private and hidden. This cooperation can dynamic, but, in the real world, the interlocutor in a business transaction that is chosen at run time is always selected in a trusted list (registry) of actors. After all, businesses do not do business with businesses they don’t (directly or indirectly) know. In the contract-first approach of service engineering, in which the service contracts are established before service implementations, the first engineering problem is how to build effective, consistent and implementable service contracts. Contracts shall act as bodies of operation, interaction, security and quality of service requirements for the digital actors that claim to implement them.

The business rule expression and inference has been initially proposed as a friendly, flexible and modular approach of the formalization and implementation of business exigencies and constraints. A recent and remarkable result of the business rule research and engineering practice field is the OMG Semantics of Business Vocabulary and Rules (SBVR) specification [43], which is, in our opinion, the most successful proposal of a formalism for business exigencies and constraints that combines ease of comprehension and formal (logical) rigor allowing mechanical model checking and automatic transformation.

In this field, the actual research and engineering practice focus on “enterprise” vocabularies and rulebooks. The general idea is to build, at the business level, a model of the organization universe of discourse and of its relationships with the organization body of shared meanings. The enterprise vocabulary constitutes the basis on which are stated the business rules that drive and discipline the enterprise’s use cases and business processes. In our best knowledge, there is no explicit and focused attempt to use the business vocabulary and rules approach to state the service contract clauses that drive and discipline the service relationships between independent organizations. The shared service contracts vocabularies and rulebooks allow independent organizations to commit each other to implement their service relationships. If these organizations have their own “enterprise” vocabularies and rulebooks, that are internal, private and often hidden, the first “service implementation” task for each of them is to establish the relationships between the service contract vocabulary and rulebook and their own enterprise vocabulary and rulebook. Note that building an enterprise vocabulary and rulebook can be a descriptive task, close to ontology modeling, while building a service contract vocabulary and rulebook is a prescriptive one.

In the quest of the best approach to manage the mapping between contract clauses and system design, implementation and testing, the contract-first SOA style naturally meets the MDA approach [38]. The MDA initiative promotes on one side the idea that key engineering activities can be organized as modeling activities, producing layered formal models, and, on the other side, the idea that formal mappings can be established between models that allow automatic generation of model elements, machine-readable components, executable code and tests.

The MDA research field, like the SOA one, is broad and far from homogeneity. For instance, a very active research thread is the DSL (Domain Specific Language) one [21], in which specialized modeling languages are established for specific business domains, facilitating mechanical model transformation and automated code generation. Otherwise, “non specialized” UML/OCL [39][40] is the preferred notation for technical models that are independent from the implementation platform. More recently, SBVR is more and more praised as a notation for models at the “business” level.

Since 2004, we have developed a model-driven methodological framework (simpleSOAD®) for designing service contracts and implementations [29]. The service contract is fully detailed as an abstract technical model independent from the specific platforms, built with UML/OCL and standard UML profiles and meta-models such as SoaML [44] and QFTC [45]. Starting from this model, we are able to generate through model mapping and transformation machine readable elements and executable code on interoperability platforms such as SOAP and REST and implementation platforms such as JEE and .NET.

The motivations for doing research on a formulation of service contracts that is formal but understandable by business people are essentially in the fact that the SOA design and implementation style allows mastering the
growing complexity and interrelation of the digital economy and pushes further its spreading in business domains where it was absent only some years ago. The digital economy interacts with our everyday lives - for instance, consider e-healthcare services - hence its dependability and security are now critical issues. The MDA research and engineering practice have achieved advanced results on the methods and tools that allow generating machine readable elements and executable components from high level specifications, such as, for instance, a high level description of a service function using an OCL/UML notation. The weak point of this overall approach is in the validation by the business people of the service specifications, namely in the fact that high level but still technical specifications of service operations, such as a UML/OCL models, cannot be realistically understood and validated by business stakeholders. Hence, it seems to us very important that service contracts were simultaneously: (i) understandable by the business stakeholders, (ii) mechanically checkable for consistency and (iii) automatically transformable into executable code.

The contribution of this paper is in a method for formulating service contract operation clauses as business rules supported by the SBVR abstract notation and expressed in constrained natural English language. Roughly speaking, the service operation is what the provider does for or on behalf of the consumer. The service operation definition is the “core” of the service contract and is the focus of this paper: it should be established independently from the interaction between the provider and the consumer that is intended to coordinate the activities of the service transaction (invocation, deliberation, delivery/use, reporting of the service operation). The service security and quality of service exigencies and constraints are orthogonal aspects of the service definition as well.

Our method of service operation definition at the business level is inspired by the contract-based design approach - which is also known as Design by contract™ [33]. This approach is already used for the service operation modeling at the technical level, as OCL proposes contract-based design elements as language traits. A service operation is described through: (i) an operation signature, a collection of vocabulary entries that refer to the service operation, the operation arguments and the operation results and that are defined by means of the entries that describe the business service activity, (ii) a collections of pre-conditions, logical statements on the state preceding the operation execution and on the values of the operation arguments, that are expressed as business operative rules; (iii) a collection of post-conditions, logical statements on the state following the operation execution, on its relationships with the state preceding it and on the result of the operation execution, which are expressed as business operative rules too.

In order to build a sound service contract vocabulary, which is the basis for an effective formulation of the pre/post-condition rules, we suggest following the methodological principles of pragmatism and parsimony. Pragmatism means that we do not look for the general meaning or the collection of meanings of the terms used in the service community universe of discourse, but only for their specific use in the “service game”, and that the contract vocabulary should register only one of those meanings, and the most specific one. Parsimony means that only (and all) the signifiers that are necessary for the formulation of the pre/post-condition rules should be registered as contract vocabulary entries. For the treatment of invariants we propose a meta-requirement inspired by the “Common Sense Law of Inertia” [56] that states that all that does not change as an explicitly intended effect of the service operation execution should stay the same. The fundamental assumptions of our method act as general requirements, such as robustness, atomicity, isolation, durability, accuracy and safety, on every service operation implementation.

The remainder of this paper is organized as follows. Section 2 details the statement of the problem that our contribution aims to solve and its background in the fields of contract-first approach of service design, implementation and testing, and of model-driven engineering. Section 3 introduces the related work in the research fields of contract-based design, business vocabularies and rules and model driven engineering. Section 4 presents the guidelines and templates that allow defining service operations through vocabulary entries and pre/post-condition operative rules with the help of a detailed example. Section 5 evokes the main methodological principles, such as pragmatism and parsimony, that should be applied to contract vocabulary building and clarifies the management of invariant properties and states and the general assumptions of the proposed method. Section 6 discusses the major advantages and drawbacks, lists future work that can be done using the results reported in this paper and briefly outlines some ideas in the requirement-based and model-based testing research field that the authors intend to develop further.

2. PROBLEM STATEMENT: HOW TO EXPRESS SERVICE OPERATION CONTRACT CLAUSES IN A MODEL-DRIVEN ENGINEERING FRAMEWORK

Contract-first service orientation recommends that each service should be precisely defined by an artifact - the service contract - that should be considered as a bundle of rights and duties by the service parties. The service contract should be established and agreed before system implementations and should act as a collection of functional and non-functional requirements. The contract-first approach is based on a limited number of distinctive assumptions:
• The basic brick of the contract-first approach is the atomic service. In an atomic service, a well defined 
  service operation must be implemented by the system acting as the provider and its results can be used 
  by the system acting as the consumer.

• An atomic service is defined by an atomic service contract that specifies, in addition to the service 
  operation, the provider and consumer interfaces and their external behaviors, and eventually includes 
  security and quality of service aspects.

• An atomic service contract doesn’t impose any functional burden to the service consumer. Moreover: 
  “... The eventual consumers of the service ... may demonstrate uses of the service beyond the scope 
  originally conceived by the provider...” [44].

• Atomic contracts can be recursively aggregated in compound contracts that formalize compound 
  services, i.e. more complex service exchanges. Generally speaking, a compound contract disciplines an 
  exchange of services between the parties whose contractual roles are the recursive combinations of the 
  component atomic contracts’ provider and consumer parts.

• The component services of a compound contract can be either simply juxtaposed - they can be invoked, 
  undertaken, declined, delivered/used and reported independently of each other - or choreographed - a 
  choreography, defined at the compound contract level, specifies the chaining and interleaving of the 
  component services invocations, deliberations, deliveries/uses, reports [10].

• Each service (atomic or compound) must have a well defined scope. A service contract does not include 
  any information or specification of operations, interfaces and behaviors outside the scope of the service.

• A service contract does not include any information or specification of internal structures, internal 
  behaviors and internal implementation technologies of the systems that enact the contract roles.

• A service contract must be practicable: someone who is aware of the contract shall be able to decide 
  whether it is being honored when she/he observes actual parties’ interfaces and behaviors. Hence, 
  someone that is aware of the contract should be able to guess how to implement a system that fulfills a 
  contract role.

• A service contract must be enforceable - someone who is aware of the contract and observes actual 
  contract parties’ interfaces and behaviors shall be able to detect and identify their discrepancies with the 
  contractual roles’ interfaces and behaviors. Hence, someone that is aware of the contract should be able 
  to guess how to test a system that claims to enact a contract role.

According to the separation of concerns stance, atomic (and compound) service contract clauses are organized 
into four relatively independent sections: (i) the operation section describes what the provider is intended to 
realize for (or on behalf of) the consumer; (ii) the interaction section describes how the provider and the 
consumer interact in order to carry out the invocation, deliberation, delivery/use and reporting of the service 
performance; (iii) the security section describes the security exigencies and constraints on the service interaction; 
(iv) the quality of service section describes the non-functional - performance, reliability, availability ... - 
exigencies and constraints on the service interaction.

In a well-designed contract the four section discipline orthogonal aspects of the service relationship that can be 
conceptually organized in a hierarchy. At the root, the operation clauses, that are independent from the 
interaction, security and quality of service clauses. The same service operation can be invoked, undertaken, 
delivered, executed and reported through different conversation protocols and, consequently, the invocation, 
deliberation, delivery/use and reporting of the service performance can be organized by different 
choreographies. Furthermore, the same service interaction can be submitted to completely dissimilar security 
and quality of service exigencies and constraints. In this paper we focus on the expression of the service contract 
“core”: the service operation clauses.

The purpose of a service operation is to deliver a business functionality as a service. Hence, the service operation 
should be firstly defined at the business level. This means that its description must be formulated in the service 
universe of discourse, i.e. by using nouns, verbs, names, signs that the service business community employs in 
their talking and writing in the course of doing the service business. Note that the service community is usually 
scattered all over independent business organizations.

The service operation definition is not a descriptive but a prescriptive task: we are not building a general 
dictionary of the service universe of discourse, we are writing a service contract. In the same way as in the “real” 
business contracts, a meaning should be given to each employed term that is well-defined, precise and 
unambiguous in the scope of the contract, if possible by choosing one of the meanings that are signified in the 
service universe of discourse and the most specific one. The collection of employed terms and definitions, that 
shall be understandable by all the service parties, constitutes the vocabulary that must be included in the contract 
as the basis for the formulation of the service clauses.

The first problem that we face is how to formulate service contracts clauses - in particular service operation 
clauses - in a form that can be easily understood and validated by the business stakeholders.

Business functionalities provided as services can be situated in very complex domains, and this complexity can be 
found in the service contract clauses too. The consistency and completeness of the contract clauses should be
firstly ensured by thorough peer review, but the availability of tools able to mechanically check some properties of the contract clauses, such as consistency, could be very helpful. This means that the language and notation used in the contract clauses should be formal. These two exigencies - formality and understandability - upon the contract clause language are conflicting. Natural language seems to satisfy the exigency of understandability, even if it is “naturally” ambiguous - after all “real” business contracts are written in natural language - but natural language statements are difficult to check for consistency. Conversely, statements in formal languages, such as UML/OCL and OWL, can be easily checked by mechanical means, but it is completely unrealistic to believe that they can be really understood by the business stakeholders.

Service contracts not only should be agreed by the business stakeholders and checked for formal properties, but also should effectively act as functional and non-functional requirements for system implementations. The Model Driven Architecture (MDA) [38] initiative is a software engineering approach that aims to represent requirements as formal models at different levels of abstraction and to map these models to system implementations [38]. The MDA approach distinguishes between: (i) the Computation Independent Model (CIM) - a declarative model of the business; (ii) the Platform Independent Model (PIM) - an abstract computational model of the system that is independent from the implementation platforms; (iii) the Platform Specific Model (PSM) - an implementable model of the system on a specific implementation platform. A strong driver of the MDA initiative is that the same system PIM should be implemented on different platforms by means of dedicated system PSMs, and the implemented systems should be functionally equivalents.

simpleSOAD® is a methodological framework based on the MDA approach that has been developed by Simple Engineering [29]. Since its inception in 2004, simpleSOAD® puts in practice a model-driven engineering cycle aimed at supporting the design, implementation and test of services and services architectures, on the basis of the OMG standards, and has been employed in successful projects in manufacturing, transportation and government. While the MDA mainstream is focused on system engineering, our methodology applies the MDA approach to service contracts that are organized as service layered models.

We have focused our past research and practice on the service PIM, which is now a structured, detailed and complete model covering all the operation, interaction, security and quality of service aspects of the service. The service PIM is built with profiled UML/OCL [57] and standard UML profiles and meta-models such as SoaML [44] and QFTC [45]. The service PSMs are implementable models of the systems’ structures and behaviors in terms of elements and functionalities provided by the underlying technological platforms. Our strict approach of the service orientation requires the service PSM level to be halved into two independent sub-levels: (i) the service interoperability PSM and (ii) the service implementation PSM. The service interoperability PSM is a model of the service interfaces and external behaviors in terms of elements and functionalities of an interoperability platform (SOAP, REST, ...), i.e. a technological platform whose purpose is to allow the interoperability of systems that are built on non-interoperable implementation platforms. The choice of the interoperability platform(s) is a contractual issue, therefore the service interoperability PSM(s) is (are) included in the service contract and consequently shared between the parties. In compliance with the contract-first stance, the interoperability PSMs are not aware of the models (CIM, PIM and PSM), if any, of the systems that claim to play a role in the service relationship [3]. These system models are private to each party that has full jurisdiction on them. In particular, the service implementation PSM is, for each service party, a model of the software components (consumer proxies, provider skeletons) upon the party’s specific implementation platform (JEE, .NET, C++, PHP, Python ...). These components provide the connection between the party’s service role instantiated upon the interoperability platform, and its internal components. In order to facilitate service implementation, our methodology proposes also a large library of consumer proxy and provider skeleton patterns for different system architectures and implementation platforms (JEE, .NET, C++, ...) [57]. Of course, no service implementation PSM is included in the contract.

In our past practice, the service CIM was an informal model, i.e. a collection of business specifications written in natural language. In compliance with the earlier versions of MDA, it was called the service Business Model. Of course, the mapping and transformation, in the MDA sense, of the informal service Business Model to the service PIM and PSM was impossible. If we try to design a service contract artifact that is both formal and understandable by the business stakeholders and we want to adopt the MDA approach in order to give it the operative role of body of requirements for system implementations, its place in the model-driven architecture is naturally at the CIM level. The third challenge of this research is how to put in place an understandable and formal service contract artifact as a service CIM that can be mapped to service PIM and PSMs.

3. RELATED WORK ON CONTRACT-BASED DESIGN, BUSINESS RULES AND MODEL-DRIVEN SERVICE ENGINEERING

3.1. Contract-based design
Design by contract™ is an approach for designing software that prescribes that software designers should define not only syntactic interface specifications for software components, but also semantic ones that extend the ordinary definition of abstract data types with pre-conditions, post-conditions and invariants. The term ‘Design by contract™’ - which is a registered trademark of Eiffel Software in the United States - and the approach were introduced by B. Meyer in connection with his design of the Eiffel programming language [32][33]. The term ‘contract’ is used in accordance with a metaphor with the conditions and obligations of business contracts. In order to avoid the confusion between the general design approach and the specific Eiffel provisions and mechanisms, we will employ in this paper the generic term ‘contract-based design’ instead of ‘Design by contract™’.

Contract-based design was firstly proposed as a paradigm and mechanism for building reliable relationships between software modules. If a software module provides a functionality, it may: (i) expect a certain condition to be guaranteed on entry by any client module that calls it - the pre-condition; (ii) guarantee a certain property on exit - the post-condition; (iii) maintain a certain property, assumed on entry and guaranteed on exit - the invariant.

Contract-based design is now broadly used as a software design approach, especially in object oriented programming, and several programming languages and environments supply contract-based design constructs and tools that generate code for checking pre/post-conditions and invariants. A step forward has been taken with the integration of the contract-based design paradigm as a specification tool in the Object Constraint Language - that is now an integrated part of the UML standard [40].

In the Web service engineering area, an early attempt to extend the WSDL standard with contract related semantic annotations (WSDL-S) [1] has produced a “light” specification (SAWSDL) [67] which is agnostic about the design approach and the semantic description language. In the service computing research domain, ideas and methods of contract-based design have been mainly used for service testing improvement. Henkel and Lohmann [18] propose to enrich service specifications - data types and operation signatures - in order to include behavioral information, such as pre/post-conditions of operations and to visualize this kind of information by graph transformation rules that blend well with a UML-based notion of data models and signatures. The operational interpretation of rules could turn out to be useful for simulating the behavior of required components in unit testing. More recently Noikajana and Suwannasart [34] describe a Web service contract with the Web Service Semantics Language (WSDL-S) and the Object Constraint Language (OCL). WSDL-S can be used to identify pre/post-conditions of Web Service operation by referring OCL expressions. The paper presents an approach for generating Web Service test cases using WSDL-S and OCL, with pair-wise testing as the test case generation method.

In the literature, the use of the contract-based design paradigm in service engineering is limited to a method for automatic test case generation, starting from some formal definition of pre/post-conditions, which is accessible to service architects and engineers but cannot be understood and validated by the business stakeholders.

3.2. Business rules

Rules are all around in the “real” economy, but yet for whatever reason they are seldom featured in requirements for the digital economy. So, the very first goal of the business rule research field and practice, from its inception, has been to give the rules the status of “first class” requirements and to distinguish them from other requirement artifacts such as use cases and process models. Rules are applied in use cases and inform and drive business processes, but are independent from these elements. As a consequence, rules can evolve independently of the other requirement artifacts and a platform that supports business rules application is intentionally built to accommodate continuous rule change.

Another essential feature of rules is that they should be captured and expressed in a form that business people can understand and ensure that they are correct. Note that, in this context, correctness means fidelity to subject matter experts’ verbalizations, not necessarily productivity for the business.

Constrained natural language is a formulation style that: (i) uses only words from a specified, relatively limited set of words in some defined natural language, (ii) uses only specified forms of phrase, from a relatively limited set and (iii) only uses them in specified combinations. A well-known example of constrained natural language is what air traffic controllers sometimes refer to as ‘Aviation English’ that uses a limited subset of English words and sentence forms.

Various authors have developed different approaches to expressing rules in natural language. The pioneering effort of T. Halpin, from the early 1990s [17] on natural language description of ORM models has produced a considerable body of work, especially on data constraint formulations in natural language. G. Witt describes a technique of modeling business rules as natural language statements issued from verbalization of Entity-Relationships models in 1993 [62]. R. Ross published in 2001 the first version of the language RuleSpeak [51] that is a general purpose constrained English language for business rules.
This field of research and practice has been consolidated in January 2008 with the release by the Object Management Group of the version 1.0 of the Semantics of Business Vocabulary and Business Rules (SBVR) specification [43]. Generally speaking, “A business vocabulary contains all the specialized terms, names, and fact type forms of concepts that a given organization or community uses in their talking and writing in the course of doing business.” [43]. The vocabulary is the supporting basis for expressing business rules: business rules build on fact types, and fact types build on concepts as expressed by terms. The SBVR’s interpretation of business rule encompasses the sense of ‘criteria’ for making decisions.

An additional and not less important driver of the SBVR’s rule specification is consistency with formal logics involving necessity (alethic) and obligation (deontic) claims. Alethic statements are structural rules, i.e. rules about how the business chooses to organize (i.e., ‘structure’) the things it deals with. Deontic statements are operative rules that govern the conduct of business activity.

SBVR does not mandate any particular notation(s) that must be used with the SBVR Metamodel. SBVR Structured English (presented in a SBVR specification annex), is a recommended proposal for expressing business rules with constrained natural English language, but is just one of possibly many notations that can be used to express the SBVR Metamodel, and, as a notation, is not normative in the SBVR standard. The research and engineering effort on business rules provides in our opinion a solid foundation for our proposal of expressing service contract clauses as business rules. In our best knowledge, all previous work is focused on data, enterprise and community rules that do not provide specific guidelines and templates for contract-first service specification and contract-based design.

3.3. Model-driven engineering

Model-driven engineering research on SBVR focuses on building software systems directly from vocabularies and rulebooks. The first step is transforming SBVR vocabularies and rulebooks into UML/OCL models. The early proposals for such transformations are simplified and limited in scope [23][49]. More effective developments concern transformation from SBVR into UML with advanced parsers for controlled formulations [21]. A complementary interesting approach is the reverse transformation from UML schemas to SBVR vocabularies and rulebooks, in order to verbalize, document and validate the UML schemas [8]. Its drawbacks are that: (i) it does not consider UML operations and (ii) it underestimates the actual difficulties of verbalizing modeling languages as UML.

A parallel research trend focuses on transforming SBVR specifications into Web Ontology Language OWL and Semantic Web rules R2ML [11]. Whereas the correspondence between OWL and SBVR is declared in the SBVR specification, in our best knowledge there are no established tools for transforming SBVR vocabularies into Web ontologies and SBVR rulebooks in Semantic Web rules at this time.

The Model Driven Enterprise Engineering (MDEE) methodology created by KnowGravity is one of the first efforts to apply SBVR in the development process [53][54][55]. MDEE supports the transformation from SBVR structural and operative rules to PIM Constraints, ECA (Event-Condition-Action) and CA (Condition-Action) rules. However, it should be noticed that SBVR business rules are logical rules, and the derivation of procedural ECA and CA rules implies further procedural assumptions that have no home in the CIM (and should be included in the PIM).

Another approach incorporates SBVR rules into IBM Model-driven Business Transformation process, integrating the interaction among rules, processes, and ontologies [24]. Marinos and Krause [30] uses the SBVR metamodel as a modeling language for building the so called generative information systems. The produced architecture is based on the RESTIV architectural style that focuses on resources identified by names, a fixed number of methods with known semantics to manipulate these resources, and stateless interactions between client and supplier. The architecture is capable of implementing atomic operations on resources, which are necessary but not sufficient for information system implementation.

Razavi and colleagues try to extend SBVR specification for declarative representations of business processes and compose them with business rules [50]. Reference [36] presents the VeTIS tool that is intended to support editing and transforming SBVR vocabularies and rules into UML/OCL models. The VeTIS tool is integrated with the MagicDraw UML CASE tool and also allows definition of requirements as UML use cases and modeling business processes as UML activities. VeTIS is compliant with the usual practice of designing information systems and may be combined with business rule implementation techniques. Preceding research is related in reference [35], where the possibilities of representing different types of business rules in UML models supplemented with OCL expressions were identified. Reference [9] presents a SBVR based software development process devoted for service-oriented information systems.
The research field of model-driven engineering of systems and services starting from SBVR vocabularies and rulebooks, forth to (and back from) “computation dependent” models such as UML/OCL models, is developing quickly. From the contract-first point of view, which is ours, past and ongoing projects have missed some essential issues. The contributions above either do not solve the same problem, i.e. they apply to system design rather than to service contract specification, or try to solve the service engineering problem without a clear contract-first stance, which limits the SBVR approach to mere data modeling.

4. SERVICE CONTRACT OPERATION CLAUSES AS BUSINESS RULES

4.1. The service contract vocabulary

In the SBVR approach, business vocabulary entries refer to concepts classified as object types and roles, individual concepts and fact types. We adopt in the next examples the following writing convention: (i) terms - that design noun concepts - are underlined (term), (ii) verbs - that design fact types - are in italics (verb), (iii) names - that design individual concepts - are double underlined and with the first letter capitalized (Name), and (iv) keywords - that design quantifiers, logical operators, modal operators ... - are in boldface (keyword). These fonts are also used for individual designations in the context of ordinary statements in order to note that defined concepts are being used.

For instance, the expression of operation clauses on a banking service needs the definition of object-type and role entries such as ‘account’, ‘balance’, ‘amount’, ‘customer’ and of fact-type entries such as ‘customer withdraws amount from account’, ‘account number identifies account’, ‘account has state’, ‘account has balance’ and so on. We do not detail the definitions of these vocabulary entries whose meaning can be considered intuitive.

A crucial formulation device for business rules is objectification. For instance, propositions based on the ternary fact type ‘customer withdraws amount from account’ can be objectified by adding to the service vocabulary the noun concept “withdrawal” and additional binary fact types such as ‘customer effects withdrawal’, ‘withdrawal involves account’ and ‘withdrawal takes out amount’. The relationship between the objectified fact type and the objectifying object types and fact types is established through a structural rule such as:

| It is necessary that | each withdrawal that is effected by a customer and that involves an account and that takes out an amount is an actuality that the customer withdraws the amount from the account |

Conventionally, in SBVR Structured English, structural rules start with the keyword ‘It is necessary that’, expressing alethic necessity, whereas operative rules start with the keyword ‘It is obligatory that’, expressing deontic obligation. The keyword ‘each’ expresses universal quantification. The keywords ‘a’ and ‘an’ express universal or existential quantification, depending on context based on English language rules. The keyword ‘the’ is used either with a designation to make a pronominal reference to a previous use of the same designation or as introduction of a name of an individual thing or of a definite description. The keyword ‘that’ is firstly used to introduce a restriction on things denoted by the previous designation based on facts about them. When followed by a propositional statement - the last in the above formulation - ‘that’ is used to introduce objectification. In SBVR Structured English, an actuality is a state of affairs (event, activity, situation, circumstance) that occurs in the actual world.

We presents rule templates by using the ABNF syntax, in which the template variables are inside angle brackets and use the notation presented above for designing their types. For instance, the template for objectification rules is:

| <objectification rule> ::= |
| It is necessary that | each <state of affairs> that <verb phrase> a <term> {and that <verb phrase> a <term>} * is an actuality that <propositional expression> |

| <state of affairs> <verb phrase> <term> is the template of the fact types that help to characterize the objectified activity, event, etc., such as ‘withdrawal is effected by customer’. <propositional expression> describes explicitly the activity, event, etc., on the basis of the fact type to be objectified, such as ‘customer withdraws amount from account’.

According to the contract-based design paradigm, the service operation is defined by means of a signature and a collection of pre-conditions and post-conditions. A signature is made of: (i) an operation name, (ii) an operation argument that conveys information supplied by the consumer and needed by the provider to deliver the service, and (iii) an operation result that conveys information to be delivered back to the consumer.
The signature is expressed as a group of related entries to be added to the service vocabulary. For instance, in order to specify a withdrawal operation signature, we define vocabulary entries such as: ‘withdrawal operation’, ‘argument’, ‘result’, ‘withdrawal operation has argument’, ‘withdrawal operation has result’, ‘argument specifies account number’, ‘argument specifies amount’, ‘result specifies account number’, ‘result specifies account balance’. These entries, whose intuitive meaning should permit to understand the example, are equipped with structural rules such as:

<table>
<thead>
<tr>
<th>It is necessary that</th>
</tr>
</thead>
<tbody>
<tr>
<td>each withdrawal operation has exactly one argument that specifies exactly one account number and exactly one amount and has exactly one result that specifies exactly one account number and exactly one account balance</td>
</tr>
</tbody>
</table>

In the example above, the keyword ‘exactly one’ expresses exactly-one quantification. The link between the service operation and the activity that delivers the service is established by a fact type such as ‘withdrawal operation handles withdrawal’, instance of the template:

<handle fact type> ::=<service operation> handles <handled state of affairs>

According to the contract-based design paradigm, the meaning of a service operation such as withdrawal operation in terms of a service activity such as withdrawal is given by its collection of pre/post-condition rules.

4.2. Service operation pre-conditions as operative business rules

Pre-conditions are logical expressions about the operation arguments and some states of the resources that shall be satisfied in the actual world in order to consent the execution of the service operation. Note that the pre-conditions considered here are objective pre-conditions. The subjective pre-conditions that are about the subjects authorized to invoke the service operation, are placed in the security section of the service contract and are stated as authorizations rules [52]. Pre-condition business operative rules are expressed with the help of formulations proposed by the SBVR Structured English.

For instance, we want to establish a rule that states that, if a withdrawal is handled by the withdrawal operation - it could be provoked by another event and, in this case, the pre-conditions do not necessarily hold - the following conditions must be satisfied when the operation execution takes place: the account, target of the withdrawal, (i) is identified by the account number supplied as an operation argument, (ii) is a “true” existing account, (iii) is active (its state is set to ‘active’), (iv) has a balance that is greater than the amount supplied as an operation argument (no allowed overdraft!). The operative rule expressing these pre-conditions is formulated below:

<table>
<thead>
<tr>
<th>It is obligatory that</th>
</tr>
</thead>
<tbody>
<tr>
<td>if</td>
</tr>
<tr>
<td>a withdrawal operation that has an argument that specifies an account number and that specifies an amount handles a withdrawal</td>
</tr>
<tr>
<td>then</td>
</tr>
<tr>
<td>the account that is involved in the withdrawal is identified by the account number immediately before the withdrawal occurs</td>
</tr>
<tr>
<td>and the account is in the extension of the concept ‘account’ immediately before the withdrawal occurs</td>
</tr>
<tr>
<td>and the account state of the account equals Active immediately before the withdrawal occurs</td>
</tr>
<tr>
<td>and the balance of the account is greater than the amount immediately before the withdrawal occurs</td>
</tr>
</tbody>
</table>

The keywords ‘if’ and ‘then’ respectively introduce the antecedent and consequent of a material implication. The placeholders that are introduced in the antecedent, preceded by the keywords ‘a’ or ‘an’, are considered universally quantified in the scope of the entire implication. We have already seen that an objectification uses a propositional expression to identify a state of affairs or event that can then be related to times and durations or be involved in any number of fact types that concern other states or events. With the fact type ‘state of affairs occurs immediately before state of affairs occurs’, the SBVR Structured English supports objectification by proposing a convenient mechanism that is based on the word ‘occurs’ being in the designation of a fact type after a placeholder. An implicit form of a fact type can be used that objectifies a propositional expression in the position of the placeholder and leaves out the word ‘occurs’. For instance, in the business rule above, the objectification takes place implicitly within the first conjunction formula of the consequent for the proposition.
‘the account that is involved in the withdrawal is identified by the account number’. This leaves out the word “occurs” before the following part of the formula (‘immediately before the withdrawal occurs’). Using these implicit forms allows objectification occurring implicitly without defining corresponding noun concepts for each fact type whose instances might be objectified.

The fact type ‘state of affairs, occurs immediately before state of affairs, occurs’ conveys the meaning of the well-known Allen’s “X meets Y” relation between the temporal interval X in which the state of affairs, occurs and the temporal interval Y in which the state of affairs, occurs. Roughly speaking, this means that state of affairs, precedes state of affairs, and there is no temporal duration between the two events. This meaning allows asserting, for each of the conjunction formulae of the consequent, that the condition expressed by the first objectified proposition of the formula must be satisfied when the operation execution starts. The second order fact types ‘concept has extension’ and ‘thing is in set’ (‘extension’ is a role for ‘set’) are used in the second conjunction formula of the consequent in order to state the condition that the object bound to the universally quantified ‘account’ variable exists as an instance of the quoted concept ‘account’.

Note that the first and the last conjunction formulae of the consequent are propositions about the relationships between the operation arguments and the states of the resources involved in the service activity performed by the operation, while the second and the third conjunction formulae are only about internal states.

The proposed template for operation pre-condition rules is:

<pre-condition rule> ::= 
it is obligatory that
if 
a <service operation> that has an <argument> that specifies an <argument value> [ and that specifies an <argument value>] * handles a <handled state of affairs>
then <propositional expression> immediately before the <handled state of affairs> occurs
[ and <propositional expression> immediately before the <handled state of affairs> occurs] *

<service operation>, <argument>, <result> are template variables for respectively the service operation, its argument and its result. <handled state of affairs> is a template variable for the objectified service activity. <service operation> has <argument>, <service operation> has <result>, <argument> specifies <argument value> and <result> specifies <result value> are the fact type templates that allow defining the service operation signature. <service operation> handles <handled state of affairs> is our “canonical” fact type template that is instantiated with the relationship between a specific service operation and the activity provided as a service. The template of each conjunction formula of the consequent, representing a condition before the operation execution, is: <propositional expression> immediately before the <handled state of affairs> occurs. Generally speaking, <propositional expression> can be a proposition about one or more <argument value> and one or more <term> related to the <handled state of affairs>, or a “type” proposition (the <entity-variable> is in the extension of the concept <entity-type>).

4.3. Service operation post-conditions as operative business rules

Post-conditions are logical expressions about (i) resource state transitions, i.e. relationships between the operation arguments and the states of the involved resources before and after the operation execution, and (ii) the truthfulness of information conveyed by the operation result. The post-conditions precisely define the service operation by the assertion of the effects of its execution, given the states before and the operation arguments.

For instance, we want to establish a rule that states that, if a withdrawal is handled by the withdrawal operation - it could be provoked by other events and, in this case the post-conditions do not necessarily hold - the following conditions must be satisfied immediately after the operation execution: (i) the balance of the account equals the balance immediately before the withdrawal minus the amount supplied as an operation argument; (ii) the account number returned as an operation result is the account number of the account; (iii) the balance returned as an operation result is the balance of the account. The operative rule expressing these post-conditions is formulated as below:

It is obligatory that
if 
a withdrawal operation that has an argument that specifies an account number, and that specifies an amount and that has a result that specifies an account number, and that specifies a balance, handles a withdrawal then the account involved in the withdrawal has a balance, immediately after the withdrawal occurs and the account has a balance, immediately before the withdrawal occurs
The fact type ‘state of affairs, occurs immediately after state of affairs, occurs’ conveys the meaning of the reverse of the Allen’s “X meets Y”, i.e. the temporal interval X in which the state of affairs, occurs “meets” the temporal interval Y in which the state of affairs, occurs. The first three conjunction formulae of the consequent define the effects of the execution of the withdrawal operation on the state of the resources, e.g. the balance of the account. The last four conjunction formulae of the consequent are conditions on the result truthfulness. Note that this rule states not only that the account number reported as a result is the actual account number, but also that this account number must be the same account number that was supplied as an argument of the service operation. As a consequence, this rule asserts in addition to the conditions informally sketched above, that the account number property of the account is an invariant of the withdrawal operation (see more about invariants in the next section). Note that the meaning of the Allen’s “X meets Y” relation of the fact type ‘state of affairs, occurs immediately before state of affairs, occurs’ (and the meaning of the inverse relation of the fact type ‘state of affairs, occurs immediately after state of affairs, occurs’) allows stating implicitly the atomicity, isolation and durability requirements of the service operation implementation. The service operation is realized by an activity that is “all or nothing”, does not exhibit intermediate states, is conducted as it was the only activity operating on the involved resources and has permanent effects “all things being equal”. The template for operation post-condition rules is:

\[<\text{post-condition rule}> ::= \]
\[\quad \text{It is obligatory that if} \]
\[\quad \quad a <\text{service operation}> \text{ that has an } <\text{argument}> \text{ that specifies an } <\text{argument value}> \text{ [and that specifies an } <\text{argument value}>^
\[\quad \quad \quad \text{ and has a } <\text{result}> \text{ that specifies a } <\text{result value}> \text{ [and that specifies a } <\text{result value}>^* \text{ handles a } <\text{handled state of affairs}> \]
\[\quad \quad \text{then} \]
\[\quad \quad \quad <\text{qualified propositional expression}> \text{ [and } <\text{qualified propositional expression}>^* \text{]}
\[<\text{qualified propositional expression}> ::= \]
\[\quad <\text{propositional expression}> \text{ immediately after the } <\text{handled state of affairs}> \text{ occurs} \mid \]
\[\quad <\text{propositional expression}> \text{ immediately before the } <\text{handled state of affairs}> \text{ occurs} \mid \]
\[<\text{propositional expression}> \]

5. METHODOLOGICAL PRINCIPLES AND ASSUMPTIONS

5.1. Some methodological hints

The purpose of the service contract vocabulary is to fix the body of meanings of the universe of discourse that is used by the contract clauses. A detailed presentation of the analysis and elicitation methods that can be applied to contract modeling at the CIM level is outside the scope of this paper, but we can sketch two attitudes that put into practice two main methodological principles: pragmatism and parsimony.

The pragmatic attitude can be concisely depicted by means of the maxim “Don’t ask for the meaning, ask for the use” [65]. The business analyst must register the terms of the universe of discourse of the service community, but he/she shall not look for the general meaning of these terms but for their specific usages in the ‘service game’, i.e. in the service business practice.

For instance, the definition of ‘withdrawal’ of the Longman Dictionary of Contemporary English (on-line) is: “

1. the act of moving an army, weapons etc away from the area where they were fighting
2. the removal or stopping of something such as support, an offer, or a service
3. the act of taking money from a bank account, or the amount you take out
4. the act of no longer taking part in an activity or being a member of an organization
5. the period after someone has given up a drug that they were dependent on, and the unpleasant mental and physical effects that this causes
6. the act of saying that something you previously said was in fact untrue [= retraction]”

When establishing the service vocabulary entry ‘withdrawal’, only the first part of the third definition above should be quoted, followed by the entry source reference as the content of the ‘Dictionary Basis’ entry caption. Of course, the other definitions do not concern the banking context, and it is easy to exclude them from the
service contract vocabulary. Conversely, the second part of the third definition (‘or the amount you take out’) reports a meaning of the term ‘withdrawal’ as a shortcut for ‘withdrawal’s amount’ that can be largely used in the service community universe of discourse, but must be excluded by the service contract vocabulary, which is prescriptive, not descriptive. We shall consent only one definition for each entry of an atomic contract vocabulary, and the most specific one. Compound contracts must manage eventual homonyms coming from component contracts, in a manner that is outside the scope of this paper. Note that the content of the ‘Definition’ entry caption could be completely formal and the dictionary basis should not be interpreted as an adopted definition. The ‘withdrawal’ and ‘Longman’ vocabulary entries look like:

<table>
<thead>
<tr>
<th>withdrawal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Definition: <strong>actuality</strong> that a given customer withdraws a given amount from a given account</td>
</tr>
<tr>
<td>Dictionary basis: ‘the act of taking money from a bank account’ <strong>Longman</strong> ['withdrawal’ 3]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Longman</th>
</tr>
</thead>
</table>

Note that, within a definition, ‘a given’ introduces an auxiliary variable into the closed projection that formalizes the definition and represents a universal quantification pushed outside of a logical formulation where ‘a given’ is used such that it represents one thing at a time – this is used to avoid ambiguity where the ‘a’ by itself could otherwise be interpreted as an existential quantification.

Also the parsimonious attitude can be concisely depicted by means of a well-known maxim - the Occam’s razor: “Entities are not to be multiplied beyond necessity” [47]. In order to obtain a body of shared meanings as simple as possible (but not simpler!) the Occam’s razor can be reinforced by: “If a sign is not necessary then it is meaningless. That is the meaning of Occam’s razor.” [64]. Our experience allows us to warn that a service contract vocabulary entry that is not used in the service activity scope is not only technically meaningless, but also harmful as a source of misunderstanding and confusion for analysts, architects and implementers. The service contract vocabulary should be the most concise one that allows the parties to fully express and understand all the operative rules that act as contract clauses.

In our experience, the strict application of these two principles leads to the construction of a service contract vocabulary that: (i) is necessary and sufficient to express the contract operation clauses as operative rules; (ii) is really shared and fully understood by a service business community that spans several organizations with no hierarchical relationships between them; (iii) is “shallow” and “on the edge”, i.e. has no ambition to register all the richness and the inevitable ambiguities of the service community universe of discourse or to grasp the complexity of one or more parties’ enterprise universes of discourse and bodies of meanings. Its only purpose is to define precisely the terms and fact types that allow formulating the service contract business rules.

The service contract vocabulary construction is a two steps task. A first version of the service contract vocabulary is built through interviews, document analysis and protocol analysis [13] and is verified by peer review. In the second step, the vocabulary is made sustainable, i.e. is augmented with new sustainable entries and the structural rules that institute the relationships and constraints between the sustainable entries and the other entries. Sustainable vocabulary entries are issued by: (i) n-ary (n > 2) fact types objectifications - the noun concept corresponding to the objectified proposition and the related binary fact types are the new sustainable signifiers; (ii) partitioning of categorization schemes - a segmentation (categorization scheme whose contained categories are complete - total - and disjoint) is defined for each categorization scheme that is not a priori complete and disjoint with respect to the general concept.

An important directive for service vocabulary sustainability is that each object type that is not a lexical object should be equipped with a complete reference scheme, i.e. with a chosen way of identifying every instance of the concept by way of related things that are either lexical or are otherwise identifiable. For instance, we define that ‘account number’, which is a role fulfilled by a lexical object, is a complete reference scheme for ‘account’, through the fact type ‘account number identifies account’, equipped with the structural rule hosted in the Necessity caption:

| account number identifies account |
|...|
| Necessity: each account is identified by exactly one account number and each account number identifies at most one account |
5.2. Invariants and assumptions

This approach of the service operation definition allows handling the well-known frame problem [25] by adopting the “Common Sense Law of Inertia” [56]. In our approach, the post-condition fully defines the operational semantics of the service operation. Only authorized state changes are expressed in the post-conditions as effects of the operation execution, and conditions about the subsistence of states of the service world that shall not change as an effect of the service operation execution (unintended side effects) are considered to be implicitly asserted. In other words, there is no need of explicitly stating invariants as “dumb” business rules such as:

It is obligatory that
if
a withdrawal operation handles a withdrawal
then
the account that is involved in the withdrawal and that is in the extension of the concept ‘account’ immediately before the withdrawal occurs is in the extension of the concept ‘account’ immediately after the withdrawal occurs
and the account number that identifies the account immediately before the withdrawal occurs identifies the account immediately after the withdrawal occurs
and the account state that is of the account immediately before the withdrawal occurs is of the account immediately after the withdrawal occurs
and ...

The absence of explicit invariant business rules is justified as the result of a general requirement of safety of the service operation implementations: state changes that are not declared as obligatory through the post-condition rules are prohibited. This meta-rule affirmed by our method is in some ways similar to the “dark world” meta-rule that is commonly adopted when formulating authorization rules: actions that are not explicitly permitted are forbidden [52].

Note that the second conjunction formula of the consequent is redundant, since it is already stated in the post-condition rule. If we consider that the safety general requirement holds, we can reformulate a more concise post-condition rule that strictly disciplines only the state changes and the truthfulness of the result:

It is obligatory that
if
a withdrawal operation that has an argument that specifies an amount and that has a result that specifies an account number and that specifies a balance handles a withdrawal
then
the account involved in the withdrawal has a balance, immediately before the withdrawal occurs
and the account has a balance immediately after the withdrawal occurs
and the balance equals the balance minus the amount
and the account number identifies the account immediately after the withdrawal occurs
and the balance equals the balance

Note that this rule affirms neither that the account number is the same immediately before and immediately after the withdrawal, nor that the account is always an instance of ‘account’ immediately after the withdrawal, but only disciplines the balance change that is the effect of the withdrawal (state change) and asserts that the account number and the balance reported through the operation result are the actual (immediately after) values of the corresponding properties of the object target of the withdrawal (result veracity). The condition that the result
account number equals the argument account number is a joint consequence of the result veracity stated in the post-condition and of the general safety requirement.

We can resume the fundamental assumptions of our contract based approach that act as general requirements on every service operation implementation. The service operation must be realized by an activity that exhibits the characteristics listed below:

- **Robustness** - the activity can be performed only if the service operation pre-condition rule is satisfied.
- **Atomicity** - the activity is “all or nothing” and does not show intermediate states - only the initial and the final ones are accessible.
- **Isolation** - the activity is conducted as the only one that accesses the resources that are referred to in the service operation pre/post-condition rules.
- **Durability** - the activity effects are permanent “all things being equal”.
- **Accuracy** - the activity effects - including the production of the service operation results - satisfy the service operation post-condition rule.
- **Safety** - the activity does not produce any effect that is not defined through the service operation post-condition rule.

6. CONCLUSION AND FUTURE WORK

The guidelines and templates presented in this paper organize in a contract-first, model-driven approach the service operation definition by means of the expression of the operation signature as a collection of vocabulary entries and of the operation pre/post-conditions as operative business rules. The main advantages of the proposed method are that: (i) pre/post-condition operative rules, that define the operation and the conditions of its application, are expressed in constrained natural language on the basis of well defined vocabulary entries, therefore they can be understood and validated by all the business stakeholders; (ii) the pre/post-condition rules are compliant with the SBVR abstract syntax, hence some of their formal properties, such as consistency, can be mechanically verified; (iii) the compliance with the SBVR abstract syntax allows the mechanical transformation of the service vocabularies and rulebooks into models represented with other formalisms - such as UML/OCL and OWL - and the automatic generation of machine-readable elements, executable code and tests.

The guidelines and templates presented above are the combined result of a research project and of practical experiences within customers’ projects. In our (limited) experimentations we have observed a level of understanding by the business stakeholders of the service contract vocabulary and rulebook that is surprisingly high, in any case much higher than the level of understanding of any other more “technical” formulation based on UML/OCL and OWL. It is a reality that SBVR still lacks of powerful editors and model checkers, but even in this situation we think that it is the best conceptual tool for expressing functional clauses of the service contract at the Computation Independent Model level.

Our future work will address different challenges that are, however, strongly related: (i) the editing tools and the formal checkers that help formulating the vocabulary entries and the rules; (ii) the tools supporting the automatic transformation of the service contract vocabulary and rulebook into artifacts based on PIM level notations, such as UML/OCL and eventually OWL; (iii) the automatic transformation of elements of the service PIM (UML/OCL, OWL) into the interoperability PSM (SOAP, REST platforms) and the implementation PSM (JEE, .NET platforms) - this issue has already been broadly investigated in the research field and by the authors, and henceforth can be considered an engineering rather than a research task; (iv) the support to the automatic generation of service samples - instantiates examples of the service interaction - starting from the service contract vocabulary and rulebook; (v) the subsequent automatic generation of requirement-based tests - test cases, oracles and contexts allowing the execution of the test cases - allowing black-box and grey-box testing of service implementations.

In our opinion, the last two challenges are among the most important and urgent. Not only service contracts should be understood and validated by business people, but also the conformance of the implementations with the contracts should be easily verified. In the digital economy, businesses do not have direct access to their business partners’ implementations, therefore the stakeholders’ trust in the implementations’ compliance with the agreed service contracts can be increased and confirmed only through the availability of appropriate contract-based testing frameworks. As the ability to perform security audits is systematically included in the security exigencies, easy testability of service implementations will be a constitutive part of their conformance criteria. We think that in a near future, it will be unthinkable to publish services that are not supported by an available service testing framework and platform.

An important feature that allows effective requirement-based testing of services is the automatic generation of executable service operation tests (test cases, oracles and contexts) from service contract vocabularies and rulebooks. This generation can pass through intermediate notations such as UML/OCL and OWL. **Operational accuracy tests** verify that implementations produce the resource state transitions and the operation result values that satisfy the service operation post-conditions, whilst **operational robustness tests** verify that implementations
Automated service testing is accomplished through the automated generation, deployment and execution of TTCN-3 components that allow to cross-check service tests with involved resource states. This will be a major feature of what we refer to as testable service implementations. We request the systematic implementation by each party of an adapter for Read, Locate and Update Services (RLUS) [42], the generic CRUD service specified by the OMG for SOAP platforms, or its companion on REST platforms, OMG hData RESTful Transport [41]. The implementation of these standard services allows easy test context loading and resource state retrieving on the System Under Test (SUT).

An accuracy cross-checking test run chains the invocations - by a test component of the automated testing platform - and for each participant SUT of: (i) the loading on the SUT of a test context that satisfies all the pre-conditions allowing for the execution of the subsequent test case; (ii) the retrieving of the SUT resource states involved in the pre-conditions evaluation in order to confirm that the test context has been correctly loaded; (iii) the submission of the accuracy test case on the SUT, whose oracle checks the matching of the SUT response with the expected one; (iv) the retrieving of the SUT resource states involved in the post-condition evaluation, in order to verify the post-condition satisfaction. The general idea is that successful cross-checking between the service operation results conveyed in the SUT response and the corresponding retrieved resource states involved in the post-condition greatly decreases the probability of remaining hidden faults of the service operation implementation.

Robustness cross-checking is based on a similar approach, with the difference that the test context selectively violates one or more preconditions of the following test case and the oracle checks that the operation invocation has been correctly declined.

A step further will be accomplished with the support of the automatic generation of safety tests, i.e. tests that aim to discover unintended side effects of the service operation implementations, even of the accurate and robust ones. The idea is to apply past artificial intelligence research results [56] to the automatic generation of hidden invariant rules from the service vocabulary and rule book (an example of such “dumb” rules is presented in the preceding section), and, starting from these “hidden” rules to enrich cross-checking tests with the retrieving of the states - before and after the service operation execution/refusal - of the resources of the SUT that shall not change as an effect of the service performance.

Automated service testing is accomplished through the automated generation, deployment and execution of TTCN-3 components [14][59][61] on an appropriate test automation platform [58] and on the automatic binding of the test platform with the Services Architecture Under Test (SAUT). The research on automated generation of test suites and environments from service operation requirements and the automated execution and evaluation of black box and grey-box testing of services architectures has already been started within the BN4SAT project conducted in partnership by the Laboratoire d’Informatique de Paris VI (LIP6) and Simple Engineering France (and partially funded by the French Association Nationale de la Recherche et de la Technologie). The project main goal is the application of probabilistic inference to the automatic scheduling of test runs on a services architecture [27][28]. The research threads depicted above will be developed in a broader project (Model and Inference Driven - Automated testing of Services architectures - MIDAS), conducted with European academic, research and industrial partners, partially funded by the European Commission within the Seventh Framework Programme, whose final objective is a testing framework and a cloud platform for Testing As A Service of services architectures, that automate all the relevant testing tasks such as test generation (of functional, security and usage-based tests), configuration of the testing environment, binding of the SAUT, test execution, test result evaluation and intelligent planning and scheduling of test campaigns.

REFERENCES


Draft - The final version will be published at ICSSEA 2012 17/18


