A Context-dependent Service Model

Naseem Ibrahim¹*, Vangular Alagar², Mubarak Mohammed²
¹Albany State University, Albany, GA, USA
²Concordia University, Montreal, QC, Canada

Abstract

In service-oriented systems a service invariably is bound to a contract. This contract includes the functionalities and quality of services guarantees that the provider can make. But such guarantees are not absolute. A service cannot guarantee its contract in all situations. It can only guarantee its contract in a predefined set of conditions. These conditions are usually related to the context of the service provider and requester. Yet, most of service-oriented applications use only service functionality as the basis of providing services and building system compositions. To remedy this situation, in this article both functionality and contract of a service are integrated into a single concept, called ConfiguredService, and formalized as a higher-order data type. The service part that includes the functionality, nonfunctional properties, service parameters, and data of the service requester, is loosely coupled to the contract part that includes trustworthiness claims, legal and business rules governing the service provision, and the context information pertaining to the provider and receiver. This loose coupling allows the creation of many ConfiguredServices, which share the same functionality but possess different contract parts. To facilitate dynamic service adaptation, we introduce a syntax and semantics for extending or modifying a ConfiguredService.

Received on 08 October 2014; accepted on 29 October 2014; published on 16 December 2014

Keywords: ConfiguredService Model, Context-dependence, Trustworthy Services, Composition Methods, Formal Verification

Copyright © 2014 Naseem Ibrahim et al., licensed to ICST. This is an open access article distributed under the terms of the Creative Commons Attribution license (http://creativecommons.org/licenses/by/3.0/), which permits unlimited use, distribution and reproduction in any medium so long as the original work is properly cited.

do:10.4108/casa.1.2.e3

1. Introduction

Service-Oriented Computing (SOC) [22] is a promising computing paradigm that uses service as the fundamental element for a rapid, low-cost development of large-scale distributed service applications in heterogeneous environments. From the recent reports [23][1] it is evident that SOC paradigm is the dominant development paradigm that is adopted by small and medium businesses, as well as many large business enterprises such as Amazon AppStore, Google, and Microsoft Market place.

An architectural model of SOC in which service is a first class element is called Service-Oriented Architecture (SOA) [19]. The main activities in SOA are service publication, service discovery and service provision. Service publication refers to the process of defining service contracts by service providers and publishing them through available service registries. Service discovery is the process of finding services that have been previously published and that meet the requirements of a service requester [39]. Service provision refers to the process of executing a selected service.

In SOA, a service provider defines the contract that can be guaranteed by a service. This contract includes the functionalities and quality of services guarantees that the provider can make. But such guarantees are not absolute. A service cannot guarantee its contract in all situations. It can only guarantee its contract in a predefined set of conditions. These conditions are usually related to the context of the service provider and requester. Legal rules also play a crucial role in constraining the publication and discovery of services. For example, a wireless phone provider may include in the service contract a guarantee of excellent quality, but this guarantee is not absolute. It may have a constraining condition stating that in order to ensure excellent quality, the consumer should be located within 1000 meters from cell phone stations. This constraint is related to the contextual information of the service consumer. In addition, local legal rules may black-out wireless service in secure-critical locations. Such legal rules should be an essential part of every contract.

Almost all current approaches use only functional and nonfunctional properties to enable the publication, discovery
and provision of services. Failure to include contextual information and legal rules will only mislead the consumer to believe in the advertised excellent quality of wireless service, regardless of where the consumer is domiciled which is not true.

Typical application domains where service-based systems should offer high quality of service are Health Care, Power Distribution, On-line Banking, On-line Shopping, and On-line Education. Services offered by these systems must be rich, trustworthy, and context-dependent in order to expand their customer base and add economic value. In this article we describe a rich, trustworthy, context-dependent service model, called ConfiguredService. We show how the service functionality and its properties, service contract, and service context are structured in a ConfiguredService model and develop a formal representation of it.

1.1. Setting for Our Work

In SOC literature many informal definitions for the term “service” exist, most of them inspired by business and telecommunication service domains. Broy et al. [31] have defined a service as a partial function and component as a total behavior. This definition implies that a component can offer finitely many services, and a service is created by the interaction of many components. They view SOA as a collection of services (functions) and their compositions. In our previous research [36, 38], we took the trustworthiness definition of Avizienis et al. [7, 8] as a compound property consisting of safety, security, reliability, and availability, and have discussed a method for formally developing trustworthy component-based systems. This method adds non-functional and trustworthiness specifications to the functional specification of Broy et al. [31]. For simplicity, we can have a component implementation with one external interface, one service, and one contract. Alternatively, we can realize a service as an interaction of many components when a component implementation has many interfaces, where an interface provides one service bound to a contract at its interface. We defined the specifications of the trustworthiness features in the component contract when the component has one interface, and in the interface contract when the component has many interfaces. We used UPPAAL [9] to formally and incrementally verify the trustworthiness of component compositions. That is, our previous research [36, 38] has orchestrated a trustworthy Service Creation Layer (SCL), constructed as a composition of trustworthy subsystems that are formally verifiable. The definition of ConfiguredService in this article is anchored on SCL.

Services in typical service domains, such as Health Care, Power Distribution, On-line Banking, On-line Shopping, involve heterogeneous object types. Consequently, services in such domains are not as easy to describe as services in small embedded systems. We need a rich service definition. We reckon that large service-oriented systems must be built in a modular fashion in order to tackle the complexity arising from the creation, deployment, and delivery of complex services in rich application domains. Layered architectures, as those explained in [12, 25], are effective ways to break the complexity barrier and promote component-based development across all layers. The layered architecture that we follow in our research is shown Figure 1. Each layer is intended to perform activities that are related to one set of related requirements of the service-centric system. SCL is accessible to every service provider (SP) and the trusted authority (TA). A SP creates trustworthy services at SCL. That is, a SP provides implementation for components which implements service functionality respecting the trustworthiness contract. The TA should verify that the implementation provided by a SP respects the trustworthiness claims of the SP. Once verified, the TA certifies the service for publication in Service Provision Layer (SPL).

In a service-oriented application, service is a first class element. Moreover the application must be user-centric in order that users may browse, query, retrieve, negotiate, and get services without any regard to how such services were created. Based upon the service descriptions, users must be...
able to compare services that have the same functionality and select the service that best suits their expectations. To suit this objective, in our research we have designed SPL, the Service Publication Layer, in which ConfiguredServices are published. In order to publish a ConfiguredService in SPL, a SP first creates a trustworthy functional implementation at SCL and gets it certified by the TA. Next, the SP creates one or more ConfiguredServices by including the certified functional implementation and adding different contract types with it. These ConfiguredServices are communicated to the TA who publishes the ConfiguredServices after verifying that the claims included in them are identical to those that have been verified earlier. Thus, by varying contracts to each certified functional implementation the SP configures many ConfiguredServices. The SP can create new ConfiguredServices at SPL by composing certified ConfiguredServices. The structure of a composite ConfiguredService is the same as the structure of a simple ConfiguredService. Hence, ConfiguredServices published in SPL are ready to be queried, browsed, compared, communicated and transacted by service requesters. Moreover, ConfiguredServices in SPL are first class objects in the system and SPL design is user-centric.

In this article we focus only on SPL and explain the structural and semantic significance of ConfiguredService, compositions of ConfiguredServices, and the production of flexible contracts in order to make the system both service-centric and user-centric. Service Request Layer (SRL) is where a service requester (SR) interacts to get services. We do not discuss this aspect in this paper. An extensive discussion on Service registry structure, query types, and service ranking are explained in [26]. Consequently, we are demonstrating that formalism can effectively be used through all the layers in developing a service-oriented system.

1.2. Contributions and Significance

Our contributions are structured as follows.

- **Context and Trustworthiness Concepts**: In Section 2 we explain the two basic concepts “context” and “trustworthiness”. Their formal representations are justified.

- **Service Model**: In Section 3 we informally explain the structure and semantics of ConfiguredService model, and illustrate it with an example chosen from car rental service domain. A formal notation for ConfiguredService and a semantics based on the formalism are given in Section 3.2. In Section 3.3 the formalism is applied to the car rental example explained earlier. In Section 3.4 we introduce a formal syntax for extension and enrichment of ConfiguredServices. The purpose of this notation is to make precise service modifications during service negotiation, and reduce communication complexity between SP and SR at service negotiation time.

Another significant component of this paper is the critical survey and comparison of related work reported in Section 4. In Section 5 we conclude our paper highlighting our contributions, and list several ongoing research activities. We use many acronyms in this paper. Table 1, which lists these acronyms, will provide a ready reference point for the reader.

2. Basic Concepts of Contract

Service functionality is created by a SP at SCL. In SPL, contract part is added by SP to the service functionality to produce a ConfiguredService. Trustworthiness, context, and legal rules are the three parts that make up a service contract. In the following sections we make precise the meanings of the terms “context” and “trustworthiness”. Legal rules are specified in a logic of context, as explained in Section 3.1.

2.1. Context

Context is a rich concept, and its implicit meaning, derived from the Latin words con (meaning “together”) and têtere (meaning “to weave”), is “weaving together”. There exists a large body of literature on context, and many definitions proposed by different researchers can be found in [10]. The survey article [43] gives a history of context, as studied


**Table 1. List of Acronyms**

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Meaning</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOC</td>
<td>Service-Oriented Computing</td>
<td>1.1</td>
</tr>
<tr>
<td>SOA</td>
<td>Service-oriented Architecture</td>
<td>1.1</td>
</tr>
<tr>
<td>SCL</td>
<td>Service Creation Layer</td>
<td>1.1</td>
</tr>
<tr>
<td>SP</td>
<td>Service provider</td>
<td>1.1</td>
</tr>
<tr>
<td>TA</td>
<td>Trusted authority</td>
<td>1.1</td>
</tr>
<tr>
<td>SPL</td>
<td>Service Provision Layer</td>
<td>1.1</td>
</tr>
<tr>
<td>SRL</td>
<td>Service Request Layer</td>
<td>1.1</td>
</tr>
<tr>
<td>SR</td>
<td>Service requester</td>
<td>1.1</td>
</tr>
<tr>
<td>SDL</td>
<td>Service Delivery Layer</td>
<td>1.1</td>
</tr>
<tr>
<td>SC</td>
<td>Service Context</td>
<td>2.1</td>
</tr>
<tr>
<td>SRC</td>
<td>Service Requester Context</td>
<td>2.1</td>
</tr>
<tr>
<td>SPC</td>
<td>Service provider Context</td>
<td>2.1</td>
</tr>
<tr>
<td>ST</td>
<td>Service Trust</td>
<td>2.2</td>
</tr>
<tr>
<td>PT</td>
<td>Provider Trust</td>
<td>2.2</td>
</tr>
<tr>
<td>SLA</td>
<td>Service Level Agreement</td>
<td>3.1</td>
</tr>
<tr>
<td>SSR</td>
<td>Service Selection Rules</td>
<td>3.1</td>
</tr>
<tr>
<td>SDR</td>
<td>Service Delivery Rules</td>
<td>3.1</td>
</tr>
<tr>
<td>SER</td>
<td>Service Execution Rules</td>
<td>3.1</td>
</tr>
<tr>
<td>SXR</td>
<td>Service Exception Rules</td>
<td>3.1</td>
</tr>
<tr>
<td>CSL</td>
<td>ConfiguredService Specification</td>
<td>4</td>
</tr>
<tr>
<td>CDL</td>
<td>ConfiguredService Description</td>
<td>4</td>
</tr>
</tbody>
</table>

by linguists and researchers in AI, long before it entered into “context-aware” system research. In these studies, a representation for context was not necessary because it was used mainly for logical reasoning and interpretation. For context-aware computing applications Schilit et al [11] was the first to propose a set of determinants for defining contexts. For Human Computer Interaction (HCI) research an easy to understand definition of context was put forth by Dey et. al [18]. However in these early studies no formal representation of context was suggested. From these research we infer the three important determinants “where, who, and what” for defining a context. In our research we have two other essential determinants “why, when”. In principle, any number of determinants may be included in a context specification, and the only criterion for choosing them is to make context a meta-information that can qualify either data or information or an entity of interest in the system.

Within SOC we can regard context as any element that could affect the service provision and execution operations. For a specific application, contexts are to be determined by domain experts with the goal to make the system behave as intended in achieving its QoS criteria. Since the three important entities in any SOA are service, service requester (SR), and service provider (SP), and each entity is influenced by its own set of contexts there are essentially three context categories. These are Service Context (SC), Service Requester Context (SRC), and Service provider Context (SPC). Contexts in each category will have the same set of determinants, however contexts in different categories may share information.

A SRC context qualifies the status of the requester SR while requesting or receiving a service. For example, the location and time parameters characterize the context of a SR while requesting or receiving a service. A context of SRC category becomes known only when a SR accesses a ConfiguredService. A context of type SPC will qualify information on service provider source and quality of service provided by a SP. As an example, a SP may have license to provide services within 10 km of the location where SP is registered. This information will be included by the SP in SPC context. A context of type SC qualifies information that defines restrictions on requesting a service, service source, and service delivery restrictions. Examples include contexts that will filter services based on location, profile of people in a location, and time constraints for service delivery.

If we demand context-aware computing within SOC then it is necessary to treat context as a first class entity and be able to make decisions based on context rules. With this in mind we have introduced ContextInfo and ContextRule in the context part of a ConfiguredService. In ContextInfo we include SPC context that qualifies location and status parameters of the SP. A SP may be able to provide a service in many contexts, and it is not feasible to enumerate all such contexts within a ConfiguredService. So, we decided to introduce ContextRules in a ConfiguredService. The significance is that the ConfiguredService is available only in contexts in which the ContextRules are true. Consequently, validating ContextRules become necessary to determine the relevant contexts for service availability and service delivery.

**Context Representation.** We use the formal representation of Wan [42] to specify ContextInfo as a context. The notation used by Wan [42] for binding ContextInfo from many dimensions is based on relational algebra semantics. A context $c$ is represented as an aggregation of ordered pairs $(X_j, v_j)$, where $X_j$ is a dimension and $v_j$ is the tag value along that dimension. Formally, if $\text{DIM} = \{X_1, X_2, \ldots, X_n\}$ is a finite set of dimensions, and $\tau_j$ is the type associated with dimension $D_j$, then any value $v_j \in \tau_j$ can be associated with $X_j$ in a context representation. The collection of pairs is written within $[\ldots]$. That is, $[X_1 : v_1, X_2 : v_2, \ldots, X_k : v_k]$ is a context in which $k$ entities (dimensions), each associated with a value, are weaved together. An example of SPC context is $[CS : \text{airticket}, SPL : \text{Chicago}]$, where service description dimension $CS$ is associated with service name, and service provider location dimension $SPL$ is associated with the location of service provider. An example of SRC context is $[SRL : \text{Montreal}, SDT : 02/22/2013, RS : \text{Internet}, PU : \text{business}]$, where SRL, SDT, RS, and PU respectively are the dimensions for service receiver location, service delivery time, resources, and purpose. If necessary, these two contexts can be combined as $[CS : \text{airticket}, SPL : \text{Chicago}, SRL : \text{Montreal}, SDT : \text{02/22/2013}]$.
02/22/2013, RS : Internet, PU : business], provided there is a ContextRule that allows a SP in Chicago to sell an air ticket for a SR in Montreal.

The inclusion of ContextInfo and ContextRule in a ConfiguredService has two additional advantages. First, we can formally validate claims encoded as a ContextRules in different contexts. Thus in our model, the logical evaluation of legal rules and trustworthiness claims can be automated. Second, the notation of context to represent ContextInfo can be viewed as an abstract data type, and can be imported as first class citizen in programming languages and in system design.

2.2. Trustworthiness

Trust adds economic value in a service transaction. The two kinds of trust included in a ConfiguredService are Service Trust (ST) and Provider Trust (PT). The ST specification should be formally verifiable by the SR, and the PT specification should be verifiable by the TA. There is no common consensus on defining PT attributes. We use “trust recommendations” from peer groups as PT attributes.

It is imperative that a user buys a ConfiguredService with the full confidence that the service included in it will perform according to ST attributes listed in it. ST specification should faithfully translate the trustworthiness features of the service created in SCL layer. Towards this we used the formal component-based approach [36] in SCL to develop trustworthy services. The development cycle is based on the process model [37] introduced by Mohammad and Alagar. This is a goal-based model in which the dependability criteria is specified at domain level and the system is developed to satisfy the criteria. That is, we brought in domain engineering as the first step in engineering services because trustworthiness criteria are specific to a domain. Moving down from the domain level to the design level, the trustworthiness criteria defined at the domain level are refined to a trustworthiness criteria for the design artifacts by selectively and incrementally adding design level properties. The evidence is provided by a formal proof or verification or some convincing manual activity to establish that (1) the trustworthiness criteria arrived at the design level is satisfied by the design principles (conflict-free completeness), and (2) the trustworthiness criteria arrived at the design level implies the domain level trustworthiness criteria (consistency). Our approach conforms to the recommendation in the report [28], where Jackson has argued that goal-based development approach should be preferred to process-based development for developing dependable systems. We provide sufficient evidence for dependability through formal verification. Consequently, the ST attributes listed in a ConfiguredService are formally verifiable. Thus, users can verify the ST claims and be convinced that the service given by the black box specification in a ConfiguredService will behave as claimed by a SP.

3. ConfiguredService Description

In this section, first we informally introduce the Configured-Service concept and explain the rationale for its structure. Next, we formalize ConfiguredService, provide its semantics, and illustrate these concepts with an example.

3.1. Structure of ConfiguredServices

The definition of ConfiguredService captures ‘what a service is’ and ‘what requirements are to be met for providing it in different contexts’. Figure 2 shows the structure of a ConfiguredService, consisting of the two parts Service and Contract. In its service part is loosely coupled to its contract part, so that the same service functionality may satisfy different contracts in different contexts.

Functionality of service, data related to service, service attributes, and nonfunctional properties are grouped under ‘service’. Thus the functional behavior of the service defined in terms of pre and post conditions do not change. Personal data of a service requester may be included in this part in order to personalize the ConfiguredService, once the client selects the service for delivery. Service requester data will be used in the validation of pre and post conditions.

The ‘contract’ part in a ConfiguredService, although roughly resembles service level agreement (SLA) in Web Services, is more expressive. It is endowed with a much richer structure in order that legal rules, trustworthiness claims, and context information that bind the service can be stated in a structured manner. Trustworthiness properties include ST and PT specifications. The former is a compound property of safety, security, reliability, and availability guarantees of the service. It is imported from SCL. PT is a statement on peer and client recommendations on the service provider. Since trustworthiness properties will significantly influence the consumer’s intent to buy the service they must satisfy the contexts associated with customer groups. Motivated by this need we decided to include context within contract part. Just by changing the contract part we can create many different ConfiguredServices, all providing the same service functionality, to suit different contexts of service delivery. As an example, providing a wireless Internet connection that costs 5$ per hour is a single service. This service might be associated with one contract stating that the quality of reception is excellent, provided the service requester is located within 50 meters from the base station. The same service may be associated with another contract stating that the quality of reception is good, provided the service requester is located beyond 50 meters but within 100 meters from the station base. Thus, we have two ConfiguredServices, each providing Internet service but with different contracts. Alternatively, it is possible to have one ConfiguredService (providing Internet service) in which more than one quality claim is included, where each claim is bound to a context rule. However, when such a ConfiguredService is personalized to a service requester, the contract will include only one quality factor, namely the one that is valid for the context of service.
requester. So, we decided to create and publish different ConfiguredServices, each specifying a set of quality claims that are valid for the context information included in the contract.

We emphasize the significance behind the separation of nonfunctional properties from trustworthiness properties in ConfiguredService model. The nonfunctional properties listed in ‘service part’ represent static quantifiable information for a service. An example is the service cost. It may be argued that service cost might vary for different service requester groups, and so cost being ‘dynamic’ should be part of the contract section. Our response to this argument is that the cost itself is fixed, however under some exceptions discounts might be offered. These arise from business policies which are changeable. Typical examples include “offering discount to senior citizens and students with authenticated ID cards”. They can be coded as business rules, which constrain the price with respect to age. If the base cost changes, or in general some of the attributes change, but the service functionality itself is not changed, then a new ConfiguredService is created using the syntax introduced in Section 3.4.

In general, a legal rule included in the Legal Issues section of Contract is a business rule constraining service availability and service delivery. The set of legal rules in our model are classified as follows:

1. (SSR: Service Selection Rules) The rules in the set SSR are necessary to validate service selection by a client.

2. (SDR: Service Delivery Rules) The rules in the set SDR should be validated at the moment the selected service is delivered to a client.

3. (SER: Service Execution Rules) The rules in the set SER are to be applied during service execution.

4. (SXR: Service Exception Rules) The rules in this set are to be applied only when abnormal situations arise.

Sets SSR and SDR must be non-empty. Sets SER and SXR may be empty for certain service domains. Service requester information, service requester context, service attributes, and service provider context are used to validate rules in the set SSR. Each rule in the set SDR must evaluate to true in the service context and service requester context at service delivery time. If SER ≠ ∅, then every rule in it is to be enforced during service execution contexts. If SXR ≠ ∅, then each rule in it specifies a reaction to be triggered when an exception arises. Typical rules of this type are those that govern contract termination before service delivery and contract violation rules during or after service execution. An essential difference between SER and SXR rules is that a SXR rule may never be fired while every SER rule will be enforced.

We emphasize that every legal rule is a ContextRule, in the sense that it includes context information either implicitly or explicitly. By restricting to predicate logic and bringing in context-dependence we are in effect using a logic of context as the semantic basis for Legal Rules in a contract. This approach follows the earlier works of McCarthy [35] and many others reviewed by Ackman [2]. In Section 3.2 we make precise the logic of context as a semantic basis for contract evaluation.

We use the context notation explained in Section 2.1 to define contexts in the Contract section, and express a context rule as a predicate logic expression. For example, the rule VERYWARM = (Temp ≥ 28) ∧ (Humid ≥ 67) might be used to determine whether or not to provide air conditioning service. The rule VERYWARM is true in infinity of contexts and false in another infinite set of contexts. As examples, VERYWARM is true in context [TEMP : 28, HUMID : 70] and it is false in context [TEMP : 26, HUMID : 65]. Therefore, the system must validate a context rule in every context of its operational cycle in order...
that it may decide whether or not service is to be provided. Thus, context rules are statements in the logic of context.

The abstract model in Figure 2 is represented both as a table, for browsing during service discovery, and as an XML file, for communicating services between different system components. The table structure of a Car-Rental ConfiguredService is shown in Table 2. The structure of this table is in one-to-one correspondence with the ConfiguredService structure hierarchy shown in Figure 2.

3.2. Formalism and Semantics of ConfiguredService

In this section we give an informal semantics for the elements in the ’service’ and ’contract’ parts of a ConfiguredService, their formal notations, and suggest a formal semantics based on it. The information content in different sections of Car Rental example in Table 2 should be understood with reference to this semantics.

Let TYPE denote a set of types. A type $T \in$ TYPE is the carrier set of data elements. Let $\mathbb{N}$ denote the universe of names used as identifiers. Attribute names $\mathbb{AN}$, variables $\mathbb{VN}$, and other entity names $\mathbb{EN}$ are disjoint subsets of $\mathbb{N}$. The function $\phi : \mathbb{N} \rightarrow$ TYPE assigns a unique type $\phi(n) = T_n$ for $n \in \mathbb{N}$. A type is uniquely associated with a set of values and operators on it. That is, for every $n \in \mathbb{N}$, $\phi_n = (\mathbb{VAL}_n, \mathbb{OP}_n)$, and for $n' \in \mathbb{N}$, $n \neq n'$, $\mathbb{VAL}_{n'} \neq \mathbb{VAL}_n$ and $\mathbb{OP}_{n'} \neq \mathbb{OP}_n$.

Semantics of Service Part. The Service part has the three sections Functionality, Nonfunctional properties and Attributes. The service functionality has been pre-created by the SP and a client can only use it as is. That is, a client has no direct access to function implementation. As such, the functionality section includes only the function precondition and postcondition. The precondition must be made true by the SR and the postcondition will be made true by the SP. A non-functional property is a constraint on the manner in which the service is delivered. Pricing information, which can itself be a complex property expressing different prices for different amount of buying, is an example of nonfunctional property. For some types of services, such as video downloading, the amount of storage required and speed of downloading may be included as nonfunctional properties. A set of Attributes is listed in the ‘Data’ section. Every attribute is typed and is represented as a name-value pair. Attributes provide sufficient information to describe a service. The service, when delivered to a client, should have these attributes. At service selection time the information in the Service part is augmented with SR information in order to personalize the service for the SR.

Formalism of Service Function

Formally, a service description in a ConfiguredService is a 3-tuple $\sigma = \langle f, \kappa, \alpha \rangle$, where $f$ is the service function, $\kappa$ is the set of nonfunctional properties, and $\alpha$ is the set of service attributes. The set $\alpha$ of service attributes is formalized as below.

$$\alpha = \{(n, v) \mid (n \in \mathbb{AN}) \land (v \in \mathbb{VAL}_n)\}$$

A non-functional property may be multi-dimensional, involving many constraints on one aspect. We should make each property atomic, in the sense that it involves one aspect only. With this perspective we can formalize the set of atomic non-functional properties as below.

$$\kappa = \{(n, v) \mid (n \in \mathbb{EN}) \land (v \in \mathbb{VAL}_n)\}$$

The function $f$ has been pre-created by the SP to provide the service that has the features specified in the sets $\alpha$ and $\kappa$. The postcondition $\text{pr}$ becomes true only after successful service provision. The precondition $\text{pr}$ should be made true, either by the SP or by the SR, in order to make the function available. Information collected from the SR prior to service personalization should be sufficient to make the precondition true. As such, $\text{pr}$ can be evaluated only after the client agrees to buy the ConfiguredService.

Semantics of Contract. The Contract part is divided into the three sections Trustworthiness, Legal Issues and Context.

1. Trustworthiness: The trust that consumer groups have on the service provider, is specified in section ProviderTrust (PT). Although there is no agreed upon definition for PT we allow the inclusion of any verifiable trust recommendations of users and peers. Service trust (ST) section enumerates safety, security, availability, and reliability claims of the service provider. Safety means timeliness guarantee and an assurance that no damage will happen during service execution. In Table 2 the statement “automatic seat belt alarms” means that an alarm rings and disables the ignition if seat belt is not worn before attempting to start the engine, and the keyword “ABS” is used to indicate that the car is equipped with Anti-lock Braking System which prevents a car from skidding while driving in hazardous conditions. Security is a composite of data integrity, authenticity, and confidentiality properties. Data integrity is concerned with the techniques to ensure the correctness of data after communication. Confidentiality is concerned with the privacy of user information. Authentication refers to verifiable client identity. These three virtues are inherent in finger-print locking system. The term “auto shut” refers the mechanism whereby the car doors are automatically locked once the car starts moving. Such a feature may be regarded either as a safety property or security property. Availability and reliability [36] are defined in terms of failures and repairs. A failure is defined as a deviation from the correct service behavior. A repair is defined as a change from incorrect service to correct service. Hence, availability is specified as the maximum wait time until the service returns back.
### Table 2. Car Rental Example

<table>
<thead>
<tr>
<th>Service</th>
<th>Function:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Name: Rent_Car</td>
</tr>
<tr>
<td></td>
<td>Pre: valid(credit_card) ∧ valid(driving_license)</td>
</tr>
<tr>
<td></td>
<td>Post: Confirm ∧ Deliver</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data</th>
<th>Product Type Data:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Size: Compact; Passenger Capacity: 5</td>
</tr>
<tr>
<td></td>
<td>Number of Doors: 4; Luggage Capacity: 4</td>
</tr>
<tr>
<td></td>
<td>Fuel Tank: 35 gallon</td>
</tr>
<tr>
<td></td>
<td>Provider Data: Company Name: U-Rent-A-Truck</td>
</tr>
</tbody>
</table>

| Non-Functional: | Rental Cost: 35$ per day |

| Trust Attributes: | Product and Service Trust (ST) |
|                  | Safety: automatic seat belt alarms, ABS |
|                  | Security: fingerprint locking, auto shut |
|                  | Reliability: no breakdown record |
|                  | Availability: guaranteed availability of car size |
|                  | Provider Trust (PT) |
|                  | Client Recommendation: 4/5 |
|                  | Organizational Recommendation: (AAA, highly recommended) |

| Contract: | SSR: [Collision and Liability insurance: client insurance] |
|          | SDR: [Rental Duration of Vehicle: 30 days maximum, Rentable Duration: 15% for AAA members and Military Personnel] |
|          | SER: [Driving Violations: renter pays before returning the car, Return of Vehicle: must be returned to rental location, Fuel: gas tank must be full at return time, Driving Range: Inside the state of rental] |
|          | SR: Service Exception Rules: [...] |

<table>
<thead>
<tr>
<th>Legal:</th>
<th>ContextInfo:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Service Provider Context: [Loc: Toronto]</td>
</tr>
<tr>
<td></td>
<td>Service Delivery Context: [Date:&lt;data of contract&gt;, Time:&lt;time of rental&gt;]</td>
</tr>
<tr>
<td></td>
<td>ContextRule:</td>
</tr>
<tr>
<td></td>
<td>SSR: [Consumer Related: age ≥ 21]</td>
</tr>
<tr>
<td></td>
<td>SDR: [Delivery Related: (time of rental + 5 minutes) ≤ car-delivery-time ≤ (time of rental + 30 minutes)]</td>
</tr>
<tr>
<td></td>
<td>SER: [Return Related: car-return-time ≤ contract-termination-time + 60 minutes]</td>
</tr>
</tbody>
</table>

2. Legal issues: This section lists SSR rules that are enforced at service selection time, SDR rules that are enforced at service delivery time, SER rules that are enforced during service execution period, and SXR rules that are enforced when exceptions arise. Rules are context-dependent, in the sense the validity of a rule should be evaluated in contexts that are implicit in the statement of the rule. As an example, the SSR rule in Table 2 is to be evaluated at the SR context which includes the profile of the SR, after the precondition of the service function evaluates to true. The SSR evaluation consists of validating the insurance certificate of the SR on the date of car rental.

In general, a legal rule is a formal statement of either a business policy to be enforced in certain contexts, or a trade law imposed by the governments who have jurisdiction over the regions where the business is conducted. Such rules, in their full generality, are often complex and require a “legal language” to express precisely. As an example, the business rule “an agreed
upon contract may be terminated without penalty by either SP or SR put into any of the three sets SSR, SDR, and SER. Such rules are hard to be mechanically verified, unless a full sensor network supports context-awareness activities in the service model.

3. Context: Both ContextInfo and ContextRules are specified in the contract section of contract. In Table 2, the context [LOC : Toronto] is a SP context, and [Date : dateofcontract, Time : timewindow – out] is the context structure in which tag information will be inserted when the service is personalized to a SR. For example, if the car is rented on 01/21/2014 at 9 hours the service provision context will be updated to [LOC : Toronto, DATE : 01/21/2014, TIME : 9]. We assume that a context toolkit [43] is linked to our system to assist the service model to automatically identify synonyms in dimension names and tag types. The three ContextRules specified in Table 2 are verifiable. The first context rule $\text{age} \geq 21$ should be verified at SR contexts at service selection time. This rule is verifiable when the attributes of SR are provided for service personalization. The second rule uses the two system variables $\text{check-out time}$ (meaning the instant of contract execution) and $\text{delivery time}$ (meaning the instant at which the car is delivered to the client). Since values for these variables will be known from the service delivery context the predicate can be evaluated. The third context rule is verifiable in the context of car return.

Formalism of Service Contract

We use the following additional set of notations:

- $\text{PR}$: a set of propositions in the trust domain
- $\text{PD}$: a finite set of predicates involving dimension names, variables, and constants
- $\text{CP}$: a finite set of consumer peer groups
- $\text{BO}$: a finite set of business organizations
- $\text{RR}$: a finite set of ordered numbers used for ranking
- $\text{TC}$: a finite set of typed contexts

Formally, a service contract is a 3-tuple $\rho = (\tau, \lambda, \gamma)$. The components of $\rho$ are explained below.

1. The trustworthiness part $\tau$ of contract $\rho$ is written $\tau = (tr_{SX\tau}, tr_{PT})$. The service trust $tr_{SX\tau}$ is a set of ordered pairs, where a pair lists “a trustworthiness feature”, and “claims for that feature” expressed as propositions. $tr_{SX\tau} = \{ (x, y) \mid x \in \{\text{Safety, Security, Reliability, Availability}\}, y = \{\text{prop} \mid \text{prop} \in \text{PR}\}\}$. Formally, $tr_{SX\tau}$ is a relation. We specify provider trust $tr_{PT}$, which is a set of recommendations from consumers and peer groups, as two functions (ordered pairs) $(ce, pe)$, where $ce : \text{CP} \rightarrow \text{RR}$, and $pe : \text{BO} \rightarrow \text{RR}$. That is, there can be only one recommendation from a peer group.

2. The component $\lambda$ is the set of legal rules, specified by extending Logic program notation with context rule. That is, $\lambda = \{ U : H \leftarrow B \mid U, H \in \text{PD} \}$, where $H$ is called the head (consequent) of the rule and $B$ is called the body (antecedent) of the rule, the “left arrow” $\leftarrow$ means “IF . . . THEN”, and $U$ is a “context rule (situation) expressed as a conjunction of predicates”. In general, the body of a rule can be a conjunction of one or more conditions; no disjunction is allowed in the body. The head of a rule is an action specification, expressed declaratively. Negations may appear in the body, but not in the head of a rule. The semantics of $U : H \leftarrow B$ is “for every context $c$ in which $U$ is true apply the action $H$, provided the guard $B$ can be made true by system variables, and the information included in the ConfiguredService description”. Since several contexts may satisfy $U$, the notation $U : H \leftarrow B$ is more compact and expressive. The rule $U : H \leftarrow B$ is not relevant in a context $c$ in which $U$ is either false or cannot be evaluated due to insufficient information. To evaluate $U$ in $c$ we require $U$ to be expressed as a conjunction of predicates involving dimension names (of contexts), variables whose values are either tag values of dimensions or data (of service, SR and SP) in the ConfiguredService, and constants. The evaluation steps are as follows:

- if a dimension name in $U$ is a DIMENSION in $c$ then replace it by the tag value associated with it in context $c$,
- substitute the variables in $U$ by their respective values, as provided either in the ConfiguredService or in the environment of evaluation,
- if $U$ still has dimension names or variables then $U$ cannot be evaluated in the context $c$; otherwise $U$ is now a proposition which evaluates to either true or false.

3. We specify $\gamma$ as an ordered pair $(\beta, \delta)$, where $\beta \subset \text{PD}$ is a set of context rules and $\delta \subset \text{TC}$ is a set of contexts. The set $\delta$ includes only service provider contexts, because service provision contexts are specified as context rules. Service selection situations that arises in the system must satisfy at least one context $c \in \delta$. Every rule in $\beta$ has to be evaluated in a context of the set SRC.

3.3. Car Rental Formalism

In this section we apply the formalism suggested in the previous section to a selected subset of Service and Contract
parts of car rental ConfiguredService shown in Table 2. We have chosen a small subset of a simple example in order to clearly explain the features of the service model. A more comprehensive example appears in [26].

We assume that CARTYPE, a finite enumerated type, is predefined. The rest of the types we need are NT (denoting natural numbers) and ST (string type). Thus, TYPE = {CARTYPE, NT, ST}. We assume that contexts relevant for service selection, service delivery, and service execution are constructed by the system. The information collected from the SR should be sufficient to evaluate the precondition. This information also enables the construction of SSR contexts and some of SDR contexts. Expert opinion should be sought in choosing a set of dimensions, because collectively they must be sufficient to validate rules. Some of the contexts that should be constructed for validating SSR and SDR rules in the contract section of Table 2 are shown below.

- **Context for validating Driving License (SSR)**
  \[ c_1 = [SR\_Name : , Driver\_License\_Num : , State\_Name : , Age : , Exp\_Date : ] \]

- **Context for validating Collision and Liability Insurance (SSR)**
  \[ c_2 = [SR\_Name : , Policy\_Num : , Insu\_Company : , Coverage : , Exp\_Date : ] \]

- **Context for Credit Card Validation (SSR)**
  \[ c_3 = [Card\_Holder\_Name : , Card\_Name : , Card\_Num : , Exp\_Date : ] \]

- **Context for Car Delivery (SDR)**
  \[ c_4 = [Date\_of\_Rental : , Time\_of\_Rental : , Rental\_Dur : , Contract\_Num : ] \]

- **Context for Car Return (SER)**
  \[ c_5 = [Return\_Date : , Return\_Time : , Return\_Loc : , Contract\_Number : ] \]

The tag fields corresponding to the dimensions in the above contexts should be collected at service selection and service delivery times. Below we give the formal representation for Car Rental ConfiguredService.

- **Formalizing Service Function:**
  The precondition is expressed as a conjunction of two predicates, and they can be evaluated at service selection time, using contexts \( c_1 \) and \( c_3 \). Using attributes and nonfunctional properties listed in Figure 2 the SP must ensure the validation of the postcondition. Necessarily such validations can be done only manually.

- **Formalizing Data and Nonfunctional parts:**
  \( \phi(\text{Size}) = \text{CARTYPE}, \phi(\text{CompanyName}) = \text{ST} \), and \( \phi \) maps the rest of the names in Data and Nonfunctional parts to NT.

- **Formalizing Trustworthiness part:**
  We use proposition names that are short hands for specifying the trustworthiness features. As an example, we use Seat\_Belt\_Alarm\_Exists as the proposition to assert the feature “automatic seat belt alarms” for safety feature. With this assumption we formally write

  \[ tr_{ST} = \{(\text{safety, Seat\_Belt\_Alarm\_Exists}), (\text{safety, ABS\_Exists}), (\text{Security, Finger\_Print\_Locking\_Exists}), (\text{Security, Auto\_Shut\_Exists}), \ldots\} \].

  The organizational trust recommendation \( tr_{PT} \) is the set

  \[
  \{(\text{client\_group, 4/5}), (\text{peer\_group, Highly\_Recommended})\}.
  \]

- **Formalizing Legal Part:**
  We use the proposition Approve with “commonsense semantics”. That is, it becomes true upon approval of the satisfaction of business rules.

  1. **Formalizing SSR rule “Collision and Liability Insurance”:**
     \[ U :: H \Leftarrow B, \text{ where } U = (\text{Client\_Name} = SR\_Name) \land (\text{Current\_Date} < \text{Exp\_Date}). \text{ H = Approve, and B = (State\_Liability\_Amount} \leq \text{Coverage}). \text{ The symbol \( \Leftarrow \) is the precedence operator defined on “date abstract data type”. The rule U is to be evaluated in context } c_2. \text{ To evaluate B the State\_Liability\_Amount, which is domain information, should be known.} \]

  2. **Formalizing SDR rule “Rental Validation”:**
     \[ U :: H \Leftarrow B, \text{ where } U = (\text{Rental\_Period} \leq \text{30}). \text{ H = Approve, and B = true.} \text{ The context to validate U is } c_4. \]

  3. **Formalizing SER rule “Fuel”:**
     \[ U :: H \Leftarrow B, \text{ where } U = (\text{Return\_Loc} = \text{Loc}), \text{ H = Approve, and B = (fuel = 35).} \text{ The context to validate U is } c_5. \text{ Dimension name Loc from SP context defined in Car Rental ConfiguredService is used in U to enforce that the car is returned to the same location where it was rented.} \]

- **Formalizing Context Part:**
  Contexts in ContextInfo part are already in formal notations. The SSR context rule “Consumer Related” requires the evaluation (Age \( \geq 21 \)) at SR context. We use the notation \( V \Theta c \) to mean “V is to be evaluated in context c”. With this notation, this rule is formally written as (Age \( \geq 21 \)) \Theta c_1, where the tags for the dimensions of context \( c_1 \) will become known at service selection time. The context rule “Delivery Related” is formalized as \( V \Theta c_4 \), where \( V = ((\text{time\_of\_rental} + 5) \leq \text{car\_delivery\_time} \land \text{car\_delivery\_time} < (\text{time\_of\_rental} + 30)) \). From context \( c_4 \) the tag value of dimension Return\_Time is substituted for the variable time\_of\_rental, and for the system variable car\_delivery\_time the time at which the car is delivered to the customer (which is generated by the system) is substituted and the predicate is resolved.
3.4. Extension and Enrichment

A SP might improve on certain quality features of a published ConfiguredService, without changing the functionality, and republish it with or without additional contractual obligations. It is also customary for a SR to negotiate on selected ConfiguredServices, requesting additions or deletions to certain contractual items. In both instances, the functionality of a ConfiguredService is not allowed to change. In this section we propose two templates as mediums for preparing modifications to a ConfiguredService. The syntax and semantics of each template are explained. During a negotiation both SP and SR may create modified ConfiguredServices, because both are aware of the structure of published ConfiguredServices. In giving the semantics we use the formal notation introduced earlier, although the template itself will be created in the natural language. Based on the semantics a tool might be built to automate the creation of modified ConfiguredServices for publication.

Extension. Informally, a ConfiguredService $\omega$ is an extension of a ConfiguredService $\omega_1$, if $\omega$ is realized by the addition of new attributes, nonfunctional properties, legal rules to those that are already in $\omega_1$. Notice that no new trustworthiness property is to be added. The syntax in Figure 3 is a shorthand for modification through extension. The first statement in the figure gives the name of the ConfiguredService created. The includes clause cannot be empty, and should list only one ConfiguredService. New information for Service and Contract parts should be listed for each sub-part in the extended-by clause. At least one sub-clause within extended-by clause should be non-empty.

The result is to create the new ConfiguredService $\omega$, whose description is constructed by copying the information listed under the clause extended-by into the respective parts of $\omega_1$.

Semantics of Extension: We use the formal notations introduced in Section 3.2. Let $\omega = (\sigma, \rho)$, where $\sigma = (f, \alpha, \kappa)$ and $\rho = (\tau, \lambda, \gamma)$ be the new ConfiguredService. It is created by extending $\omega_1 = (\sigma_1, \rho_1)$, where $\sigma_1 = (f_1, \alpha_1, \kappa_1)$, $\rho_1 = (\tau_1, \lambda_1, \gamma_1)$ with the information listed in Figure 3. The extension semantics is $f = f_1$, $\alpha = \alpha_1 \uplus \alpha_n$, $\kappa = \kappa_1 \uplus \kappa_n$, $\tau = \tau_1$, $\lambda = \lambda_1 \uplus \lambda_n$, $\delta = \delta_1 \uplus \delta_n$, and $\beta = \beta_1 \uplus \beta_n$, where the operators $\uplus$ and $\ast$ have the following semantics.

- **Semantics of $\uplus$**: Let $X$ and $Y$ denote two sets of ordered pairs. For an ordered pair $(u, v)$, let first $(u, v) = u$, and second $(u, v) = v$. Let dom and ran be unary operators defined on any set of ordered pairs, which respectively extract the set of first and second components from the ordered pairs in $X$. That is $\text{dom}(X) = \{x \mid (\text{first}(x), t) \in X\}$, and $\text{ran}(X) = \{x \mid (\text{second}(x), t) \in X\}$. We define $X \uplus Y = X \cup \{t \mid (t \in Y) \land ((\text{first}(t) \notin \text{dom}(X)) \land ((\text{second}(t) \notin \text{ran}(X))$.

- **Semantics of $\ast$**: Let $X$, $Y_1$ and $Y_2$ be sets of rules, where (1) the rules in each set are classified into

```
ConfiguredService $\omega$
includes ConfiguredService $\omega_1$
extended-by {
  Service
    Data Attributes $\alpha_n : \ldots$
    Nonfunctional $\kappa_n : \ldots$
  Contract
    Legal Rules $\lambda_n :$
    $\text{SSR}_n : \ldots$
    $\text{SDR}_n : \ldots$
    $\text{SER}_n : \ldots$
  Context $\gamma_n$: ContextRules $\beta_n :$
    $\text{SSR}_n : \ldots$
    $\text{SDR}_n : \ldots$
    $\text{SER}_n : \ldots$
  ContextInfo $\delta_n : \ldots$
}
```

Figure 3. ConfiguredService Extension Syntax

SSR, SDR, and SER rules, and each rule is formalized as $U :: H \Leftarrow B$. Writing $X = \{\text{SSR}, \text{SDR}, \text{SER}\}$, $Y_1 = \{\text{SSR}_1, \text{SDR}_1, \text{SER}_1\}$, and $Y_2 = \{\text{SSR}_2, \text{SDR}_2, \text{SER}_2\}$, the set $X = Y_1 \ast Y_2$ is calculated as

$$\text{SSR} = \text{SSR}_1 \cup^* \text{SSR}_2,$$
$$\text{SDR} = \text{SDR}_1 \cup^* \text{SDR}_2,$$
$$\text{SER} = \text{SER}_1 \cup^* \text{SER}_2,$$

where $\cup^*$ means set union with the inherited rules marked in the result. The applicability of an inherited rule is determined only when context information becomes known. An inherited rule that is applicable is “unmarked” and retained, whereas an inherited rule that is not applicable is removed.

Enrichment. Informally, a ConfiguredService $\omega$ is an enrichment of a ConfiguredService $\omega_1$, if $\omega$ is realized by changes to some or all of attributes, nonfunctional properties, trustworthiness properties, legal rules, and context part of $\omega_1$. The trustworthiness part is added to the syntax in Figure 3 to get the enrichment syntax shown in Figure 4. New information denoting changes to Service and Contract parts should be listed for each sub-part in the enriched-with clause. The rest of syntactic constraints are similar to the extension syntax. The result is to create the new ConfiguredService $\omega$, whose description is constructed by overwriting the information listed under the clause enriched-with into the respective parts of $\omega_1$.

Semantics of Enrichment: We use the formal notations introduced in Section 3.2. Let $\omega = (\sigma, \rho)$, where $\sigma = (f, \alpha, \kappa)$ and $\rho = (\tau, \lambda, \gamma)$ be the new ConfiguredService. It is created
by enriching \( \omega_1 = (\sigma_1, \rho_1) \), where \( \sigma_1 = (f_1, \alpha_1, \kappa_1) \), \( \rho_1 = (\tau_1, \lambda_1, \gamma_1) \) with the information listed in Figure 4. The enrichment semantics is \( f = f_1, \alpha = \alpha_1 \oplus \alpha_n, \kappa = \kappa_1 \oplus \kappa_n, \tau = \tau_1 \oplus \tau_n, \lambda = \lambda_1 \otimes \lambda_n, \delta = \delta_1 \otimes \delta_n, \) and \( \beta = \beta_1 \otimes \beta_n \) where the operators \( \oplus, \otimes, \oplus, \) and \( \otimes \) have the following semantics.

- **Semantics of \( \otimes \):** We use the functions \( \text{first} \), and \( \text{second} \) defined for an ordered pair \( t \), and functions \( \text{dom} \) and \( \text{ran} \) defined for any set \( X \) of ordered pairs. For two sets of ordered pairs \( X \) and \( Y \)
  \[
  X \oplus Y = \{ t | t \in X \land \text{first}(t) \notin \text{dom}(Y) \} \cup \
  \{ t' | t' \in Y \land \text{first}(t') \in \text{dom}(X) \}
  \]

  - **Semantics of \( \otimes \):** Let \( X, Y_1 \) and \( Y_2 \) be sets of rules, where \( (1) \) the rules in each set are classified into SSR, SDR, and SER rules, and each rule is formalized as \( U :: H \iff B \). Writing \( X = \{ \text{SSR}, \text{SDR}, \text{SER} \}, Y_1 = \{ \text{SSR}_1, \text{SDR}_1, \text{SER}_1 \}, \) and \( Y_2 = \{ \text{SSR}_2, \text{SDR}_2, \text{SER}_2 \} \), the set \( X \cup Y_2 \) is calculated as
  \[
  \text{SSR} = \text{SSR}_1 \uplus \text{SSR}_2, \\
  \text{SDR} = \text{SDR}_1 \uplus \text{SDR}_2, \\
  \text{SER} = \text{SER}_1 \uplus \text{SER}_2,
  \]

where the semantics of \( X \uplus T \) is explained below:

**Figure 4. ConfiguredService Enrichment Syntax**

**Figure 5. ConfiguredService: Multiple Modification**

1. Extract the context conditions from all rules in set \( T \):
   \[
   T_1 = \{ U | (U :: H \iff B) \in T \}
   \]
2. Filter out from set \( S \) all rules that have their context conditions in set \( T_1 \):
   \[
   S_1 = \{ U :: H \iff B | ((U :: H \iff B) \in S) \land (U \in T_1) \}
   \]
3. Replace the rules in the subset \( S_1 \) of \( S \) with rules in \( T \):
   \[
   R = (S \setminus S_1) \cup T
   \]

- **Semantics of \( \odot \):** We may regard \( \delta_1 \) and \( \delta_n \) as sets of ordered pairs. Each ordered pair is of the form \( (\text{dim}, \text{value}) \). With this view the semantics is
  \[
  X \odot Y = \bigcup_{c_i \in C, c_j \in Y} \{ (c_i \ominus c_j) \}
  \]

Table 3 shows a modification of the Car Rental ConfiguredService in its first column, and an enrichment of the Car Rental ConfiguredService in its second column.

An enriched (extended) ConfiguredService may be included in creating a new ConfiguredService by extension (enrichment). In such cases, the semantics is applied in the order of inclusion. Figure 5 shows an example for creating NewCar – Rental service by enriching ECar – Rental service which is an extension of Car – Rental service.

### 4. Related Work and Comparison

Research in SOC has produced a large volume of work ranging from pure business perspectives to pure software engineering perspectives. In order to critically evaluate and place our work in the context of the current state of the art in SOC the four dimensions context, trustworthiness, service modeling, and formalism are identified by us, because these are the cornerstones of our current work in SPL. Within this confine we have diligently chosen a significant subset of published works for comparison with our work.
Table 3. A Modified and Extended ConfiguredService

<table>
<thead>
<tr>
<th>Modified Car Rental</th>
<th>Extended Car Rental</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ConfiguredService</strong> <strong>MCar-Rental</strong></td>
<td><strong>ConfiguredService</strong> <strong>ECar-Rental</strong></td>
</tr>
<tr>
<td><strong>includes</strong> ConfiguredService Car Rental</td>
<td><strong>includes</strong> ConfiguredService Car Rental</td>
</tr>
<tr>
<td><strong>enriched-with</strong></td>
<td><strong>extended-by</strong></td>
</tr>
<tr>
<td><strong>Contract:</strong></td>
<td><strong>Service:</strong></td>
</tr>
<tr>
<td>Legal Rules:</td>
<td>Data Attributes:</td>
</tr>
<tr>
<td>SER: car return:</td>
<td>Renter Name:</td>
</tr>
<tr>
<td>(no fee when returned to the location</td>
<td>Second Driver:</td>
</tr>
<tr>
<td>where rented),</td>
<td>driver license:ALP109</td>
</tr>
<tr>
<td>(50$ additional fee if returned to</td>
<td>age: 52</td>
</tr>
<tr>
<td>another location})</td>
<td></td>
</tr>
<tr>
<td>SSR: collision and liability insurance:</td>
<td>Contract:</td>
</tr>
<tr>
<td>fully covered</td>
<td>context γ</td>
</tr>
<tr>
<td>Trustworthiness:</td>
<td>ContextRules β</td>
</tr>
<tr>
<td>tr_ST: 9/10</td>
<td>SSR: {Consumer Related: age ≤ 65}</td>
</tr>
</tbody>
</table>

4.1. Context

Context information has been used by several researchers [6, 14, 16] for service adaptation. In [41] execution context is part of service planning. These works have brought in context only for the purpose of service execution and not for service modeling. They neither have a formal representation of context, nor do they discuss service models. Below we give a detailed comparison between our context formalism and the use of contexts by others.

- In [41] the authors discuss a planning-based adaptation middleware. They mention the need to adapt ubiquitous systems under dynamically evolving run-time execution contexts. However we do not find context being used anywhere in the planning stages.

- In [6], although context is claimed to be a first class entity in the model, there is no explicit representation for it in the model, and the scope of context is limited to identifying logical and physical resources available during service provision. It is unclear how such context-dependent service provision can be achieved without context representation. In their example on video-streaming service they state that the consumer side execution context includes memory, screen resolution, and processing power. The paper does not explain how these context requirements are matched by service property.

- A context-aware composition method offered by COTS (Commercial-Off-The-Shelf) is reported in [16]. In particular they focus on composing context-aware mobile and pervasive systems, where devices and applications dynamically find and use components from their environment. They have defined a context profile, public and private context attributes, and distinguish between static and dynamic attributes. This context definition is the basis of their work for service composition and protocol compatibility.

- A model-based development on context-aware web applications is discussed in [14]. The authors have defined context-awareness as the ability of the system to use context either for delivering content or for performing system adaptations or both. This paper does not define context and service model. Instead, the authors have used the term reactive, adaptive, and context-sensitive to convey context-awareness capability.

Analysis. None of the above works have defined context formally. Moreover they have not considered service models, with their focus being only on service provision scenarios. In our work we have used the formal syntax and semantics of context, introduced by Kaiyu Wan [42], in the ConfiguredService model. Thus, context is a first class entity, and a context may be used in many ConfiguredService models. Being a distinguished member of the service model, the context in the model can be manipulated independently from the rest of the information in the service model. The context profile used in [16] loosely resembles the context representation that we use. However, there are two major differences. These are (1) the context representation that we use allow any number of distinct dimensions, and (2) the tags (values) associated with each dimension has a type, which may even be an abstract data type. Relational algebra semantics is given to the contexts that we use. This has allowed the introduction of context operators [42], thus building a rich context algebra as the basis of a context toolkit for plug-in to the development of context-aware applications. To our knowledge, we have not seen any previous work in which context is made a first class citizen of service model in a formal manner.
4.2. Trustworthiness

In almost all existing work in SOC either trustworthiness is not part of service model or is included with nonfunctional attributes in the service model. In our research we distinguish between nonfunctional attributes, which are related to service function, and trustworthiness attributes, which is part of a contract between the service provider and service requester. For us, trustworthiness is a moral value concept that should exist in the market place, because according to the Economic Theories [21] trust adds economic value. So, we built SPL on top of SCL, which is the platform for verifiably formal construction of trustworthy systems [38]. Hence, trustworthiness properties included by SP in a ConfiguredService model are verifiable using the SCL implementation. That is, a SR can assess the trustworthiness claims, either directly or through TA, before accepting a service selection.

Analysis. We have utilized the advantages of layered architecture to import trustworthy services into SPL from SCL. We have not come across any previous work that has either adapted a layered architecture or other means to include trustworthiness features within a service model. The significance of our work is that we allow users to validate the trustworthiness claims before committing to buy the service. Such a validation is context-dependent. Because the ConfiguredService model includes the service provider context and the constraints on service provision, a user by knowing her context information can validate some aspects of the service provision constraints, such as locality and time, while browsing the published service. In [33] a formal treatment of adapting Web services with regard to specific security requirements is given. It requires further research to see whether this approach might be generalized to include other trustworthiness features, such as safety and reliability, for Web services and then subsequently adapted for any service model.

4.3. Service Model

The modeling approaches can be classified based either on the language, or the architecture or a combination of both. The two main languages that have been used for modeling services are UML [15, 34], and WSDL with the related Web description languages [24, 32, 40, 45]. Architecture based service modeling approach uses an Architectural Definition Language (ADL) [17, 29] to describe services. There are a few other methods [13, 20] which combine language and some abstract architectural details for describing service features. Figure 6 compares these service models with ConfiguredService with respect to the seven attributes functional, nonfunctional and trust, legal rules, context, formalism, verification support, and tool support.

The UML-based language UML4SOA [44] supports a model-driven development of SOA architecture. No precise guidelines exist for creating such an architecture. This approach relies mainly on the intuition of the developer, and lacks formalism. The UML model is transformed into IOM (Intermediate Orchestration Model), which is transformed into PSM (Platform Specific Model). A limited repertoire of nonfunctional properties and legal rules may be stated in the model. Tool support is available for modeling and no tool exists for analysis. The family of Web Services Description Languages (WSDL) and OWL-S (including SWS) [30, 32, 45]) have been used to model services. Semantic embedding of data is enabled by SWS, however these languages are not formal. They do not provide any support for stating legal rules, and offer no verification support. In OWL-S it is possible to include some nonfunctional properties. The model has no provision for including trustworthiness properties.

The three architectural description languages SOADL [17, 29], SRML [20], and SOFM [13] provide formal notations for modeling services. The SODAL formalism uses Pi calculus, and models a service as a process. Compositions of services are done using synchronized message passing. In SRML a service module is the basic unit of design. An activity module, another process, is created to satisfy a specific requirement. Internal policies can be defined in an activity module, and they govern initialization and interaction constraints. External policies express constraints on Service Level Agreements (SLA). The Service-Oriented Feature Model (SOFM) [13] captures the service features provided by an application in an abstract form. A service feature represents the requirements of an application as a collection of services. Service features are classified into constraints and refinements. Constraints are the set of static relationships between service features. Constraints are further classified and specialized into three types of relationships. These relationships form the basis for select service features, and compositions.

In SRML, legal rules (policies) can be specified but trustworthiness properties cannot be stated. Both SOADL and SOFM provide support for specifying the nonfunctional properties but provide no support for including legal rules.

Analysis. Figure 6 shows the relative merits of these approaches. While service functionality is part of every model, nonfunctional and trustworthiness properties are not supported by two of the models and the other four models provide only partial support for including nonfunctional and trustworthiness properties. We emphasize that those models that provide such partial support do not make a distinction between nonfunctional, legal, and trustworthiness properties. Contextual information is not part of any approach. Consequently, the relationship between contract and context is totally ignored by all these approaches. A couple of approaches, although have ignored the modeling of nonfunctional and trustworthiness properties, have used formal methods and conducted formal verification. This means that the formal verification targets only the correctness of service functionality. Except for UML and Web services languages the other approaches provide only a minimum amount of tool support for modeling services.


<table>
<thead>
<tr>
<th></th>
<th>Functional</th>
<th>Nonfunctional and Trust</th>
<th>Legal Rules</th>
<th>Context</th>
<th>Formal</th>
<th>Verification Support</th>
<th>Tool Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>UML-based</td>
<td>YES</td>
<td>SOME</td>
<td>SOME</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>SRML</td>
<td>YES</td>
<td>NO</td>
<td>SOME</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>SOADL</td>
<td>YES</td>
<td>SOME</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>SOFM</td>
<td>YES</td>
<td>SOME</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>WSDL</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>OWL-S WSMO</td>
<td>YES</td>
<td>SOME</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>Configured Service</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Figure 6. Service Models Comparison

The ConfiguredService model has many merits compared to these approaches. The integration of Service from Contract, while maintaining a loose coupling, is novel. The syntax to create new ConfiguredServices through extension and modification of ConfiguredService is supported by a rigorous semantics, which offers an easy to use method for creating flexible contracts and services. We have given a formal notation to describe ConfiguredServices. Based on this formal notation we are able to provide a rigorous semantics for composing ConfiguredServices. Other modes of descriptions of ConfiguredServices are Table structure, Architectural Description Language, and XML [26]. Thus, our service model is rich, formal, flexible, and generic. Because of the set theoretic semantic basis, we can view a ConfiguredService as an abstract data type and import it to any implementation platform.

5. Conclusion

The major contribution of this paper is the formal ConfiguredService model. This contribution has remedied the lack of support in existing models for a formal comprehensive integration of nonfunctional, trustworthy, legal and contextual information in a service contract. This is achieved in the following manner.

- **Providing support for trustworthiness information:** We introduced a formal service model that considers service trust and provider trust. In service trust we specified safety, security, availability and reliability. In provider trust we specified peer recommendations and recommendations from independent organizations.

- **Binding context to the service contract:** We defined the service contract to include a formal specification of the service provider context, and situations governing service provision. The situations are context conditions that constrain the service contract.

- **Including legal rules in service definition:** We included the specification of the legal rules governing business model of the service provider within the contract definition.

We have given syntax and semantics for creating flexible ConfiguredServices. These short hand notations are useful during contract negotiations. The model-based formal notation for ConfiguredService helps towards composition formalization, as well as during later implementation stages [26]. In addition to the Table structure, Ibrahim [26] has provided two other representations. One is the ConfiguredService Specification Language (CSL) for the use of non-experts of service architecture. The other is the ConfiguredService Description Language (CDL) that is mainly intended to be used within the system for formal analysis and communication between processes.

On top of this model we have completed the following projects as part of the development of a framework for creating trustworthy services.

- A formal composition theory for ConfiguredService models and a formal proof of compliance that the contract of a composite ConfiguredService is in satisfaction with the contracts of the ConfiguredServices in the composition.

- A comparison of three high-level database models was done for implementing context and publication of ConfiguredServices. Based on this study we chose Hbase to implement context, context history and Service Registry in which ém ConfiguredServices are published. These results appear in [3, 4]. A user interface for browsing and querying has also been implemented.

- Detailed description of Service Registry, browsing, selection, and ranking of query processing of ConfiguredServices based on different types of queries have
been studied. In [27] we have reported service discovery for different query types and one ranking algorithms. Subsequently Ammar Alsaiq [5] has compared a number of ranking algorithms that are possible candidates for service ranking, and developed a new multifaceted algorithm which will rank services discovered from the Service Registry to fulfill the preferences specified by a user in the query.

We are currently developing tools to (1) project different views of published and selected services, (2) create flexible ConfiguredServices for negotiation, and (3) interactively compose ConfiguredServices based on user preferences.

References


