EAI Endorsed Transactions
on Context-Aware Systems and Applications
Research Article ICST.ORG

A Context-dependent Service Model★

Naseem Ibrahim[1,*], Vangular Alagar[2], Mubarak Mohammmed[2]
[1]Albany State University, Albany, GA, USA
[2]Concordia 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

doi: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

★Please ensure that you use the most up to date class file, available from EAI at http://doc.eai.eu/publications/transactions/latex/

*Corresponding author. Email: naseem.ibrahim@asurams.edu

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 *ConfiguredServic* 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. Alternately, 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

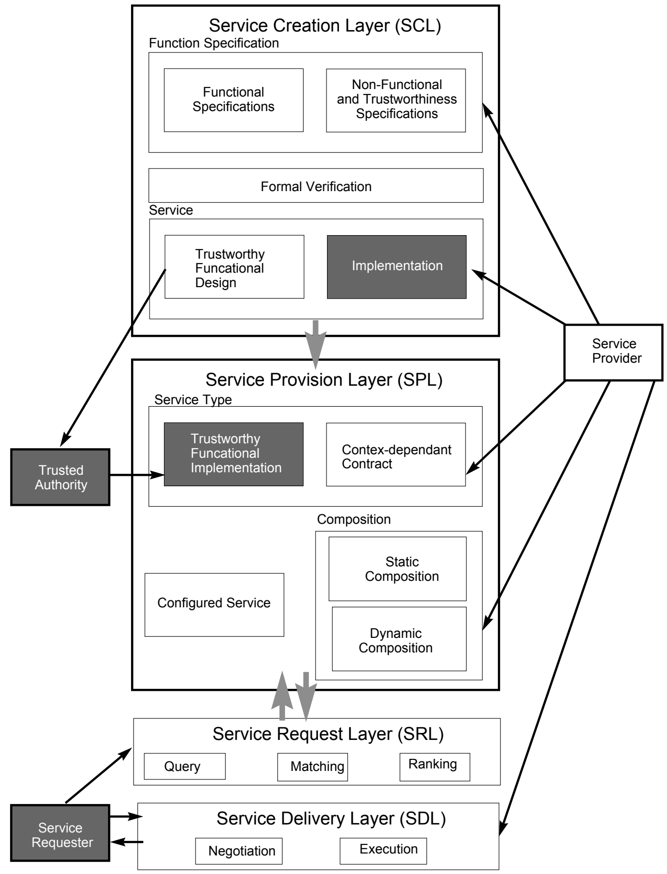

Figure 1. Layered SOA

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 algorithms are discussed in [3, 5, 26]. The primary reasons for explaining the layered SOA are to focus on the setting of the research reported in this article, and bring out the advantages of layering to *ConfiguredService* model.

- There is clear separation of goals. Each layer addresses one goal.
- Implementation details are hidden away from clients who will query and retrieve only *ConfiguredServices* from SPL. Consequently, a SP has the freedom to choose an implementation platform for SCL that best fits his goals.
- Trustworthy component-based methodology used in SCL provides many additional advantages to service development process. Component designs can be modified without affecting the ultimate result. The service function is formally verified for trustworthiness properties before it is included in a *ConfiguredService*.
- Given that a *ConfiguredService* in SPL has been certified by the TA, it is sufficient to verify contract fulfillment at service execution times.
- There is a clear formal path that goes from functional and non-functional specification to component and contract specification (SCL), to ConfiguredService specification and their compositions (SPL), and finally to service matching and ranking (SRL). SCL, as illustrated in our previous work [36, 38] has been formalized. We explain in this paper how the SPL layer can be formalized. Algorithms for service matching 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 *ConfigureService* 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 *texere* (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

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

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 the 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 $DIM = \{X_1, X_2, \ldots, X_n\}$ is a finite set of dimensions, and τ_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 [...]. 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 $[CS : airticket, SPL : Chicago]$, 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 $[SRL : Montreal, SDT : 02/22/2013, RS : Internet, PU : business]$, 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 $[CS : airticket, SPL : Chicago, SRL : Montreal, SDT :$

$02/22/2013, RS : Internet, PU : business$], 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 *ConfiguredService* 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

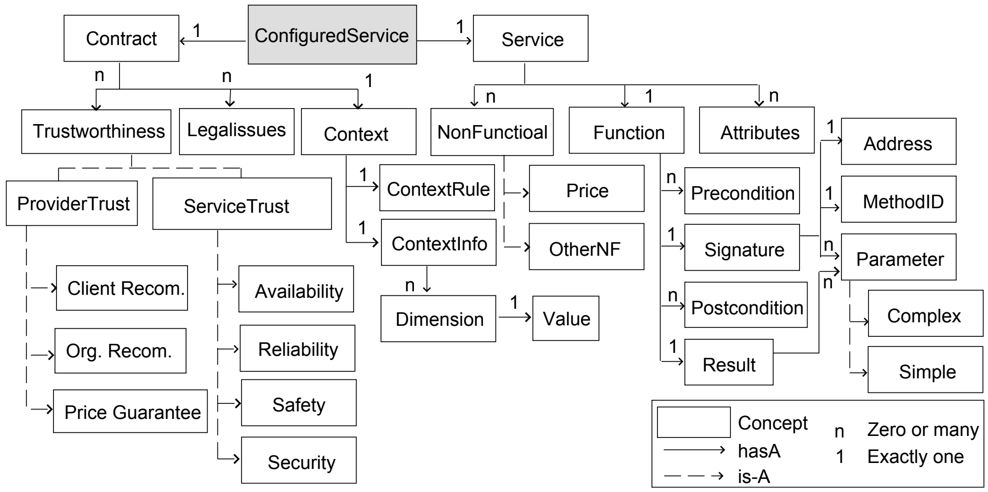

Figure 2. *ConfiguredService* Structure

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 $\neq \emptyset$, then every rule in it is to be enforced during service execution contexts. If SXR $\neq \emptyset$, 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 $VERYWARM = (Temp \geq 28) \wedge (Humid \geq 67)$ 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 $\sigma = <f, \kappa, \alpha>$, where f is the service function, κ is the set of nonfunctional properties, and α is the set of service attributes. The set α of service attributes is formalized as below.

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

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.

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

The function f has been pre-created by the SP to provide the service that has the features specified in the sets α and κ. The postcondition po becomes true only after successful service provision. The precondition 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 agreement personalization should be sufficient to make the precondition true. As such, 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

Service	Function:	*Name:* Rent_Car *Pre:* valid(credit_card) $\wedge$ valid(driving_license) *Post:* Confirm $\wedge$ Deliver
	Data	*Product Type Data:* Size: Compact; Passenger Capacity:5 Number of Doors: 4; Luggage Capacity: 4 Fuel Tank: 35 gallon *Provider Data:* Company Name: U-Rent-A-Truck
	Non-Functional:	*Rental Cost:* 35$ per day
Contract:	Trust Attributes:	*Product and Service Trust* (ST) *Safety:* automatic seat belt alarms, ABS *Security:* finger-print 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)
	Legal:	SSR: {*Collision and Liability insurance:* client insurance} SDR: {*Rental Duration of Vehicle:* 30 days maximum, *Rental Discount:* 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} SXR: Service Exception Rules: {. . . }
	Context	*ContextInfo:* Service Provider Context: [Loc : $Toronto$] Service Delivery Context: [$Date$:<data of contract>, $Time$:<time of rental>] *ContextRule:* SSR: {Consumer Related: age $\geq$ 21} SDR: {Delivery Related: (time of rental + 5 minutes) $\leq$ car-delivery-time $\leq$ (time of rental + 30 minutes)} SER: {Return Related: car-return-time $\leq$ contract-termination-time + 60 minutes}

to operate correctly, and reliability is defined as the guaranteed maximum number of failures in a unit of time.

In Table 2 the term "no breakdown record" implies maximum reliability, and the term "guaranteed availability" means availability without waiting time. The SP imports trustworthiness properties implemented in SCL to the *ConfiguredService*. Since trustworthiness properties included in the contract part have been formally verified in service implementation in SCL a SR should trust the peer reviews listed under PT in order to believe the features stated under ST. Should this not happen, the SR can invoke the services of TA (see Figure 1) in order to get the claims listed under ST verified.

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 within 48 hours from signing" in SXR cannot be 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 $[Date : <dateofcontract>, Time : <timeofcheck-out>]$ 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 $age \geq 21$ 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 *check-out-time* (meaning the instant of contract execution) and *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:

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

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

1. The trustworthiness part τ of contract ρ is written $\tau = (tr_{ST}, tr_{PT})$. The service trust tr_{ST} is a set of ordered pairs, where a pair lists "a trustworthiness feature", and "claims for that feature" expressed as propositions. $tr_{ST} = \{(x, y) \mid x \in \{Safety, Security, Reliability, Availability\}, y = \{prop \mid prop \in \mathbb{PR}\}\}$.
Formally, tr_{ST} 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 $ce : \mathbb{CP} \to \mathbb{RR}$, and $pe : \mathbb{BO} \to \mathbb{RR}$. That is, there can be only one recommendation from a peer group.

2. The component λ is the set of legal rules, specified by extending Logic program notation with context rule. That is, $\lambda = \{U : H \Leftarrow B \mid U, H \in \mathbb{PD}\}$, 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 γ as an ordered pair (β, δ), where $\beta \subset \mathbb{PD}$ is a set of context rules and $\delta \subset \mathbb{TC}$ is a set of contexts. The set δ 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 β 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 $\mathbb{NT}$ (denoting natural numbers) and $\mathbb{ST}$ (string type). Thus, $TYPE == \{CARTYPE, \mathbb{NT}, \mathbb{ST}\}$. 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) $c_1 = [SR\_Name : \quad , Driver\_License\_Num : \quad , State\_Name : \quad , Age : \quad , Exp\_Date : \quad]$
- Context for validating Collision and Liability Insurance (SSR) $c_2 = [SR\_Name : \quad , Policy\_Num : \quad , Insu\_Company : \quad , Coverage : \quad , Exp\_Date : \quad]$
- Context for Credit Card Validation (SSR) $c_3 = [Card\_Holder\_Name : \quad , Card\_Name : \quad , Card\_Num : \quad , Exp\_Date : \quad]$
- Context for Car Delivery (SDR) $c_4 = [Date\_of\_Rental : \quad , Time\_of\_Rental : \quad , Rental\_Dur : \quad , Contract\_Num : \quad]$
- Context for Car Return (SER) $c_5 = [Return\_Date : \quad , Return\_Time : \quad , Return\_Loc : \quad , Contract\_Number : \quad]$,

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

- 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 $tr_{ST} == \{(safety, Seat\_Belt\_Alarm\_Exists), (safety, ABS\_Exists), (Security, Finger\_Print\_Locking\_Exists), (Security, Auto\_Shut\_Exists), \ldots\}$.
  The organizational trust recommendation tr_{PT} is the set $\{(client\_group, 4/5), (peer\_group, Highly\_Recommended)\}$.

- *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":
  $U :: H \Leftarrow B$, where $U == (Client\_Name = SR\_Name) \wedge (Current\_Date \prec Exp\_Date)$, $H == Approve$, and $B == (State\_Liability\_Amount \leq Coverage)$. The symbol $\prec$ is the precedence operator defined on "date abstract data type". The rule U is to be evaluated in context c_2. To evaluate B the $State\_Liability\_Amount$, which is domain information, should be known.
  2. Formalizing SDR rule "Rental Validation":
  $U :: H \Leftarrow B$, where $U == (Rental\_Period \leq 30)$, $H = Approve$, and $B == True$. The context to validate U is c_4.
  3. Formalizing SER rule "Fuel":
  $U :: H \Leftarrow B$, where $U == (Return\_Loc = Loc)$, $H == Approve$, and $B == (fuel = 35)$. The context to validate U is c_5. 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 $V == ((timeofrental + 5) \leq car - delivery - time \wedge car - delivery - time \leq (timeofrental + 30))$. From context c_4 the tag value of dimension $Return\_Time$ is substituted for the variable $timeofrental$, 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* ω is an *extension* of a *ConfiguredService* ω_1, if ω is realized by the addition of new attributes, nonfunctional properties, legal rules to those that are already in ω_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* ω, whose description is constructed by *copying* the information listed under the clause **extended-by** into the respective parts of ω_1.

ConfiguredService ω
  includes *ConfiguredService* ω_1
  extended-by {
    Service
      Data Attributes $\alpha_n : \ldots$
      Nonfumctional $\kappa_n : \ldots$
    Contract
      Legal Rules λ_n :
        $SSR_n : \ldots$
        $SDR_n : \ldots$
        $SER_n : \ldots$
      Context γ_n:
        ContextRules β_n :
          $SSR_n : \ldots$
          $SDR_n : \ldots$
          $SER_n : \ldots$
        ContextInfo $\delta_n : \ldots$
  }

Figure 3. *ConfiguredService* Extension Syntax

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 $f = f_1$, $\alpha = \alpha_1 \uplus \alpha_n$, $\kappa = \kappa_1 \uplus \kappa_n$, $\tau = \tau_1$, $\lambda = \lambda_1 \circledast \lambda_n$, $\delta = \delta_1 \cup \delta_n$, and $\beta = \beta_1 \circledast \beta_n$, where the operators $\uplus$ and $\circledast$ 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 $dom(X) = \{first(t) \mid t \in X\}$, and $ran(X) = \{second(t) \mid t \in X\}$. We define $X \uplus Y = X \cup \{t \mid (t \in Y) \wedge ((first(t) \notin dom(X))\}$.

- *Semantics of* $\circledast$: 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 $U :: H \Leftarrow B$. Writing $X = \{SSR, SDR, SER\}$, $Y_1 = \{SSR_1, SDR_1, SER_1\}$, and $Y_2 = \{SSR_2, SDR_2, SER_2\}$, the set $X = Y_1 \circledast Y_2$ is calculated as

$$SSR = SSR_1 \cup^{\star} SSR_2,$$

$$SDR = SDR_1 \cup^{\star} SDR_2,$$

$$SER = SER_1 \cup^{\star} SER_2,$$

where $\cup^{\star}$ 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* ω is an *enrichment* of a *ConfiguredService* ω_1, if ω is realized by changes to some or all of attributes, nonfunctional properties, trustworthiness properties, legal rules, and context part of ω_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* ω, whose description is constructed by *overwriting* the information listed under the clause **enriched-with** into the respective parts of ω_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

ConfiguredService ω
    includes *ConfiguredService* ω_1
    enriched-with {
        Service
            Data Attributes $\alpha_n : \ldots$
            Nonfumctional $\kappa_n : \ldots$
        Contract
            Trustworthiness τ_n :
                $tr_{ST_n} : \ldots$
                $tr_{PT_n} : \ldots$
            Legal Rules λ_n :
                $SSR_n : \ldots$
                $SDR_n : \ldots$
                $SER_n : \ldots$
            Context γ_n:
                ContextRules β_n :
                    $SSR_n : \ldots$
                    $SDR_n : \ldots$
                    $SER_n : \ldots$
                ContextInfo $\delta_n : \ldots$
    }

Figure 4. *ConfiguredService* Enrichment Syntax

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 $f = f_1$, $\alpha = \alpha_1 \oplus \alpha_n$, $\kappa = \kappa_1 \oplus \kappa_n$, $\tau = \tau_1 \boxplus \tau_n$, $\lambda = \lambda_1 \otimes \lambda_n$, $\delta = \delta_1 \Cup \delta_n$, and $\beta = \beta_1 \otimes \beta_n$ where the operators $\oplus$, $\boxplus$, $\otimes$, and $\Cup$ have the following semantics.

- *Semantics of* $\oplus$: We use the functions $first$, and $second$ defined for an ordered pair t, and functions dom and ran defined for any set X of ordered pairs. For two sets of ordered pairs X and Y

$$X \oplus Y = \{t \mid t \in X \wedge first(t) \notin dom(Y)\} \cup$$
$$\{t' \mid t' \in Y \wedge first(t') \in dom(X)\}$$

- *Semantics of* $\boxplus$:

$$\tau_1 \boxplus \tau_n = (tr_{ST_1} \oplus tr_{ST_n}, tr_{ST_1} \oplus tr_{ST_n})$$

- *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 $U :: H \Leftarrow B$. Writing $X = \{SSR, SDR, SER\}$, $Y_1 = \{SSR_1, SDR_1, SER_1\}$, and $Y_2 = \{SSR_2, SDR_2, SER_2\}$, the set $X = Y_1 \otimes Y_2$ is calculated as

$$SSR = SSR_1 \dagger SSR_2,$$
$$SDR = SDR_1 \dagger SDR_2,$$
$$SER = SER_1 \dagger SER_2,$$

where the semantics of $S \dagger T$ is explained below:

ConfiguredService $NewCar - Rental$
    includes *ConfiguredService* $ECar - Rental$
    enriched-with{
        {copy the rest of MCar-Rental}
    extended-by{
        Contract:
        Legal Rules:
            SER: {Car Delivery: Free car upgrade for AAA members}
        }
    }

Figure 5. *ConfiguredService*: Multiple Modification

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

- *Semantics of* $\Cup$: We may regard δ_1 and δ_n as sets of ordered pairs. Each ordered pair is of the form $(dim, tagvalue)$, With this view the semantics is

$$X \Cup Y = \bigcup_{c_i \in C, c_j \in Y} \{(c_i \oplus c_j)\}$$

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

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

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 butprovide 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.

	Functional	Nonfunctional and Trust	Legal Rules	Context	Formal	Verification Support	Tool Support
UML-based	YES	SOME	SOME	NO	NO	NO	YES
SRML	YES	NO	SOME	NO	YES	YES	YES
SOADL	YES	SOME	NO	NO	YES	YES	YES
SOFM	YES	SOME	NO	NO	YES	YES	NO
WSDL	YES	NO	NO	NO	NO	NO	YES
OWL-S WSMO	YES	SOME	NO	NO	NO	NO	YES
Configured Service	YES	YES	YES	YES	YES	YES	YES

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 am 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 Alsaig [5] has compared a number of ranking algorithms that are possible candidates for service ranking, and developed a new multifacetted 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

[1] Abrams, C. and Schulte, R.W. (2008) *Service-Oriented Architecture Overview and Guide to* SOA *Research*. Technical report, Gartner Research, Stamford, CT.

[2] Akman, V. and Surav, M. (1996) Steps towards formalizing context. *AI Magazine* **17**(3): 55–72.

[3] Alsaig, A. (2013) *Context-Aware Service Registry: Modeling And Implementation*. Master thesis, Concordia University.

[4] Alsaig, A., Alsaig, A., Mohammad, M. and Alagar, V. (2013) Modeling and managing context and context history. In *Proceedings of the International Conference on Context-aware Systems and Applications, Ho Chi Minh City, Vietnam*.

[5] Alsaig, A. (2013) *A Semantic-based Ranking Algorithm for Services in Service-oriented Systems*. Master thesis, Concordia University.

[6] Autili, M., Cortellessa, V., Marco, A.D. and Inverardi, P. (2006) A conceptual model for adaptable context-aware services. In *International Workshop on Web Services Modeling and Testing (WS-MaTe 2006)*.

[7] Avizienis, A., Laprie, J.C. and Randell, B. (2001) *Fundamental Concepts of Dependability*. Research report n01145, laas-cnrs.

[8] Avizienis, A., Laprie, J.C., Randell, B. and Landwehr, C. (2004) Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing* **1**(1): 11–33.

[9] Behrmann, G., David, A. and Larsen, K.G. (2004) A tutorial on UPPAAL. In *Formal Methods for the Design of Real-Time Systems: 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM-RT 2004* (Springer–Verlag), **LNCS 3185**: 200–236.

[10] Bettini, C., Brdiczka, O., Henricksen, K., Indulska, J., Nicklas, D., Ranganathan, A. and Riboni, D. (2010) A survey of context modelling and reasoning techniques. *Pervasive and Mobile Computing* **6**(2): 161–180.

[11] Bill Schilit, N.A. and Want, R. (1994) Context-aware computing applications. In *Mobile Computing Systems and Applications (WMCSA1994)* (IEEE): 85–90.

[12] Broy, M. (2003) Modeling services and layered architectures. In *Formal Techniques for Networked and Distributed Systems, H. König et al. eds. Lecture Notes in Computer Science, vol. 4767* (Springer-Verlag): 48–61.

[13] Cao, X.X., Miao, H.K. and Xu, Q.G. (2008) Modeling and refining the service-oriented requirement. In *TASE '08: Proceedings of the 2008 2nd IFIP/IEEE International Symposium on Theoretical Aspects of Software Engineering* (Washington, DC, USA: IEEE Computer Society): 159–165.

[14] Ceri, S., Daniel, F., Matera, M. and Facca, F.M. (2007) Model-driven development of context-aware web applications. *ACM Transactions on Internet Technology* **7**(1).

[15] Contact), A.J.B.P. (2008) Service oriented architecture modeling language (SOAML) - specification for the UML profile and metamodel for services (UPMS), OMG Submission document: ad/2008-11-01.

[16] Cubo, J., Canal, C. and Pimentel, E. (2011) Context-aware composition and adaptation based on model transformation. *Journal of Universal Computer Science* **17**(5): 777–806.

[17] Dan, X., Shi, Y., Tao, Z., Xiang-Yang, J., Zao-Qing, L. and Jun-Feng, Y. (2006) An approach for describing soa. In *International Conference on Wireless Communications, Networking and Mobile Computing, WiCOM 2006*: 1–4.

[18] Dey, A.K., Abowd, G.D. and Salber, D. (2001) A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction* **16**(2): 97–166.

[19] Erl, T. (2007) SOA *Principles of Service Design* (Upper Saddle River, NJ, USA: Prentice Hall PTR).

[20] Fiadeiro, J.L., Lopes, A. and Bocchi, L. (2006) A formal approach to service component architecture. In Bravetti, M., Núñez, M. and Zavattaro, G. [eds.] *Web Services and Formal Methods. LNCS, vol 4184* (Springer, Berlin Heidelberg), 193–ï£¡213.

[21] Fritz, M., Hausen, T., Schefer, G. and Canavari, M. (2005) Trust and electronic commerce in the agrifood sector: a trust model and experimental experience. In *Proceedings of the XIth International Congress of the EAAE (European Association of Agricultural Economists): The Future of Rural Europe in the Global Agri-Food System.*

[22] Georgakopoulos, D. and Papazoglou, M.P. (2008) *Service-Oriented Computing* (The MIT Press).

[23] Heffner, R. (2008) *SOA adoption: Budgets donï£¡t matter much*. White paper, Forrester Research, Cambridge, MA.

[24] Herrmann, M., Aslam, M.A. and Dalferth, O. (2007) Applying semantics (wsdl, wsdl-s, owl) in service oriented architectures (soa). In *Proceedings of the 10th Intl. Protege Conference* (Budapest, Hungary).

[25] Herzberg, D. and Broy, M. (2005) Modelling layered distributed communications systems. *Formal Aspects of Computing* **17**(1): 1–18.

[26] Ibrahim, N. (2012) *Specification, Composition and Provision of Trustworthy Context-dependent Services*. Phd thesis, Concordia University.

[27] Ibrahim, N., Mohammad, M. and Alagar, V. (2013) Publishing and discovering context-dependent services. *Human Centric Computing and Information Systems* **3**: 1–22.

[28] Jackson, D. (2009) A direct path to dependable software. *Communications of the ACM* **52**: 78–889.

[29] Jia, X., Ying, S., Zhang, T., Cao, H. and Xie, D. (2007) A new architecture description language for service-oriented architecture. In *Sixth International Conference on Grid and Cooperative Computing (GCC 2007)*: 96 –103.

[30] Kashyap, V., Bussler, C. and Moran, M. (2008) *The Semantic Web, Semantics for Data and Services on the Web* (Springer).

[31] M. Broy, I.H.K. and Meisinger, M. (2007) A formal model of services. *ACM Transactions on Software Engineering*

Methodologies **16**(1): 1–40.

[32] MARTIN, D., PAOLUCCI, M., MCILRAITH, S. and ET AL, M. (2004) Bringing semantics to web services: The owl-s approach. In *First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004)* (San Diego, California, USA).

[33] MARTIN, J. and PIMENTEL, E. (2011) Contracts for security adaptation. *The Journal of Logic and Algebraic Programming* **80**: 154–179.

[34] MAYER, P., SCHROEDER, A. and KOCH, N. (2008) Mdd4soa: Model-driven service orchestration. In *EDOC '08: Proceedings of the 2008 12th International IEEE Enterprise Distributed Object Computing Conference* (Washington, DC, USA: IEEE Computer Society): 203–212. doi:http://dx.doi.org/10.1109/EDOC.2008.55.

[35] MCCARTHY, J. and LIFSCHITZ, V. (1990) *Formalizing Commonsense: Papers by John McCarthy* (Greenwood Publishing Group Inc.).

[36] MOHAMMAD, M. and ALAGAR, V. (2011) A formal approach for the specification and verification of trustworthy component-based systems. *Journal of Systems and Software* **84**: 77–104.

[37] MOHAMMAD, M. and ALAGAR, V. (2012) A component-based development process for trustworthy systems. *Journal of Software: Evolution and Process* **24**: 815–835.

[38] MOHAMMAD, M.S. (2009) *A Formal Component-based Software Engineering Approach for Developing Trustworthy Systems*. Phd thesis, Concordia University, Montreal, Canada.

[39] PAPAZOGLOU, M.P. (2008) *Web Services: Principles and Technology* (Prentice Hall), 1st ed.

[40] ROMAN, D., KELLER, U., LAUSEN, H., DE BRUIJN, J., LARA, R., STOLLBERG, M., POLLERES, A. *et al.* (2005) Web service modeling ontology. *Applied Ontology* **1**(1): 77–106.

[41] ROUVOY, R., ELIASSEN, F., FLOCH, J., HALLSTEINSEN, S. and STAV, E. (2008) Composing components and services using a planning-based adaptation middleware. In *Proceedings of the 7th international conference on Software composition*, SC'08 (Berlin, Heidelberg: Springer-Verlag): 52–67.

[42] WAN, K. (2006) *Lucx: Lucid Enriched with Context*. Phd thesis, Concordia University, Montreal, Canada.

[43] WAN, K. (2009) A brief history of context. *International Journal of Computer Science* **6**(2): 33–42.

[44] WIRSING, M., BOCCHI, L., FIADEIRO, J.L., GILMORE, S., HOELZL, M., KOCH, N., MAYER, P. *et al.* (2008) Sensoria: Engineering for Service-Oriented Overlay Computers. In DI NITTO, E., SASSEN, A.M., TRAVERSO, P. and ZWEGERS, A. [eds.] *At Your Service: Service Engineering in the Information Society Technologies Program* (MIT Press).

[45] ZAREMBA, M., KERRIGAN, M., MOCAN, A. and MORAN, M. (2006) Web services modeling ontology. In CARDOSO, J. and SHETH, A.P. [eds.] *Semantic Web Services, Processes and Applications* (Springer), 63–87.