A Formal Service Contract Model for Accountable SaaS and Cloud Services

Joe Zou
IBM Australia
Macquarie University
Sydney, NSW, Australia
joezou@au1.ibm.com

Yan Wang
Department of Computing
Macquarie University
Sydney, NSW, Australia
yanwang@science.mq.edu.au

Kwei-Jay Lin
Department of EECS
University of California
Irvine, CA 92697, USA
klin@uci.edu

***Abstract*— Enabled by Service-Oriented Architecture (SOA), recently Software as a Service (SaaS) and Cloud computing are gaining momentum in the industry. An open issue is how to ensure accountability in business services offered through Internet. Traditionally a contract is an effective legal means to uphold accountability in business transactions. In this paper, we propose a novel service contract model called OWL-SC for e-Services. Based on OWL-DL and SWRL, OWL-SC model can be used to disclose obligations of both e-Services consumer and e-Services provider. More importantly, the model allows service participants to monitor the service contract execution and keep track of obligation fulfillment for each party during service delivery. We also propose a graphical model SC-CPN based on Colored Petri-Nets (CPN) to formally model contract obligations and their interdependencies. SC-CPN can also be used to validate the correctness of obligations in OWL-SC through simulation and state space analysis. Finally, we use the Congo Book service as an example to illustrate how to use OWL-SC and SC-CPN to build a service contract model.**

Keywords: Service Contract, Accountability, Cloud, SaaS

I. INTRODUCTION

Based on the principles of SOA, SaaS and Cloud Computing are emerging as the new business models that have the potential to transform the IT industry. In contrast with traditional on-line services, SaaS and Cloud Computing turn non-trivial software and infrastructure capabilities into business services that can be massively subscribed and consumed through the internet. The viability of these business models thus depends on the service providers' reputations and more importantly, consumers' confidences on the services offered. The critical factor that underpins these reputations and confidences is the accountability of the services. By *accountability*, we mean clear disclosure of service obligations; faithfully honoring of disclosed obligations, or otherwise assuming the liability for the non-performance of the obligations.

Traditionally, accountability is achieved through the enforcement of a legal, paper-based contract. In SaaS, service providers generally publish a terms and conditions page for their offerings on their web-site. Upon clicking on the acceptance link, the consumer enters a binding contract with the service provider. Essentially, this is a "web-enabled" version of the paper-based contract, which presents serious accountability challenges in cyberspace. In its plain-text form, a web-enabled paper-based contract can neither be interpreted by software agents, nor be used as a basis for contract execution monitoring and state reasoning. Thus it does not enable service obligation disclosure, nor allows software agents to decide which party is responsible for what action, and which party is liable for what result. Moreover, the virtualized nature of the e-Services makes it even more difficult for liability settlement. This may become a major obstacle for the take up of these e-Services.

The criticality of these issues motivates the need for a formal construct that maps the obligations in a paper-based contract to machine interpretable capability statements. We call this formal construct a *service contract* model. Current SOA standards such as WS-* and REST do not provide a service contract model. Without that, the obligation monitoring and liability assignment would be baseless. This presents an accountability gap in current implementations of SOA that underpins the SaaS and Cloud platforms.

In this paper, we propose a formal *service contract* model OWL-SC to bridge the accountability gap in current SOA implementations. OWL-SC defines an ontology for service contract. It captures the obligations of service participants in a legal contract, representing them as a machine-interpretable formalism based on OWL-DL and SWRL. Such formalism facilitates obligation **disclosure**, **monitoring** and contract state **reasoning** for service participants during the full-lifecycle of service consumption. To assist in OWL-SC model development and validation, we also propose a graphical *service contract* model called SC-CPN. SC-CPN is an extension of Colored Petri-Nets (CPN), which offers strengths in both state-based and event-based modeling approaches. With SC-CPN, the behavioral aspect of a service contract can be visually modeled; the contract execution can be simulated; and the properties of the contract can be analyzed to ensure model validity.

The next section reviews related work. Section III outlines the OWL-SC model. This is followed by section IV that discusses the SC-CPN model. An example using Congo Book service to illustrate the use of OWL-SC and SC-CPN models is presented in section V. Finally, we conclude this paper and discuss our future work in section VI.

II. RELATED WORK

While the term *accountability* is frequently used in different contexts, Schedler provides a definition that succinctly captures the essence of accountability: "A is accountable to B when A is obliged to inform B about A's (past or future) actions and decisions, or justify them and to

be punished in the case of misconduct" [1]. The early work on accountability in IT focuses on certain properties such as non-repudiation, fairness of interactions, tamper-evident, etc. at a technical protocol level. Recent research begins to address accountability concerns such as quality of services (QoS), root cause analysis, autonomous recoverability, reputation and provenance at an architect level. Refer to [2] for a detailed review on Accountability literature. Overall, accountability research in IT literature mainly focuses on some technical properties and rarely addresses the accountability concerns in the underlying business contract. Such accountability concerns are obligation **disclosure** and **liability** for misconduct, as suggested in Schedler's definition. While Service Level Agreement (SLA) is a kind of contract, it normally only records the non-functional aspect of obligations and falls short on the functional aspect of service obligations, which is the key concern for business.

On the other hand, e-Contract is an extensive researched area in the IT literature. IBM's Trading Partner Agreement (TPA) defines e-contract as an XML document (TPAml) that stipulates the general contract terms, conditions, participant roles, communication and security protocols, and business process [3]. TPAml has been submitted to OASIS and used as a basis for developing ebXML Collaboration Protocol Profile (CPP) and Collaboration Protocol Agreement (CPA). TPAml and ebXML CPP/CPA are designed for business to business (B2B) process integration which requires a full stack of infrastructure support on both sides. This heavy weight approach is not suitable for e-Services, especially for SaaS or Cloud services. In [4], the author proposes a multi-party e-Contract model that maps a paper-based contract into contract actions and contract commitments. An algorithm is outlined to detect contract violation based on the commitment graph. In [5], an e-Contract model based on Modal Action Logic, Deontic Logic and Subjective Logic is presented. In [6], a Business Contract Language (BCL) and Formal Contract Language (FCL) are proposed using Defeasible Logic and Deontic Logic. Other approaches include applying Event Calculus to e-Contract (ecXML) [7] and extending First Order Logic (FOL) to handle dynamic aspect of service contract [8]. Most of these approaches use some variant of FOL to represent e-Contract, which is not easy to seamlessly integrate into SOA architecture; and moreover, most of these approaches favour expressiveness at the expense of decidability—as we can see that FOL is not decidable.

An interesting contribution on service contract based on Description Logic (DL) is presented in [9]. The authors outline a logic framework that incorporates concrete domain and action theory into an expressive DL called $\mathcal{ALCQO}(Q^*)$. The logic framework has the expressive power to describe both static information and dynamic behaviour aspects of a service contract while still remains decidability. The authors do not provide a service contract representation in their work and it is not clear how the service contract is used in an SOA environment. This logic framework may have a limitation on its reasoning power as DL has limited reasoning capability on relationships between roles.

Grosof and Poon address this problem by combining RuleML and DAML+OIL in [10]. They use DAML+OIL to represent MIT Process Handbook's process ontology and also present a contract ontology for the process. Then they outline an approach to specify RuleML rules "on top of" DAML + OIL ontology to enable specification of more complex behaviors in the contract. While their approach has more expressiveness, the implication of decidability issue was not discussed. Also how to apply the contract model in a service environment was not covered either.

While the above models make significant contributions on the topic of e-Contract, we have yet to find a model which can be easily used to enable accountability for SaaS.

III. A Pro-Accountability Service Contract Model

A. Requirements for Accountability Management

For SaaS and Cloud in particular, we can summarize the core requirements for enabling accountability as below:

AR 1: Obligations for both service provider and consumer can be specified unambiguously and interpretable by software agents.
AR 2: Obligations can be readily disclosed and accessible in a Web based environment.
AR 3: Obligations can be monitored and breaches can be immediately tracked; the status of contract execution can be reasoned by software agents.
AR 4: Evidence of obligation fulfillment can be easily examined and reported.

B. A Service Contract Model for SaaS and Cloud

In order to meet the above requirements, we propose a *service contract* model as a formal construct for SaaS. The service contract specifies obligations of both service provider and service consumer, which can be used as a basis for service participants to justify or explain their actions. Moreover, it can also be used as a baseline for obligation tracking and breach determination. It is formally defined as:

Definition 1: A service contract is a tuple $SC = (s, D, P, Op, Oc, Seq, st, R, T)$, where: s is a service offered; D is a finite set of domain specific contract term definitions; P is a pair of involved parties (provider *pr* and consumer *pc*); Op (Provider Obligation) is a finite set of (Action, Evidence) pair: $Op = \{(ap_1, ep_1), (ap_2, ep_2), \ldots, (ap_n, ep_n)\}$; Oc (Consumer Obligation) is a finite set of (Action, Evidence) pair: $Oc = \{(ac_1, ec_1), (ac_2, ec_2), \ldots, (ac_k, ec_k)\}$. In Op and Oc, Action is a tuple: $a = (input, output, pre, post)$, where $input$, $output \in D$, both pre and $post$ are binary condition expressions; Evidence is a finite set of evidence object, timestamp and condition triple: $E = \{(o_1, t_1, c_1), (o_2, t_2, c_2), \ldots, (o_n, t_n, c_n)\}$, where $o_i \in D$, t_i is the creation timestamp of the evidence object, c_i is a condition expression that is evaluated to *true*, $1 \leqslant i \leqslant n$; Seq is a finite set of sequences $Seq = \{s_1, s_2, \ldots, s_n\}$, where each s_i is a sequence of actions; Contract State $st \in S$, $S = \{st_1, st_2, \ldots, st_n\}$, where st_i is one of user defined contract states, for example, *initialisation*, *in progress*, *provider breaching* contract, etc; Rules: $R = \{ r_1, r_2, \ldots r_n \}$ is a horn clause: *consequent* ← *antecedent*; Time Period $T = \{contract\_start\_time, contract\_end\_time\}$.■

Definition 1 provides a generic two-party *service contract* structure that captures the key accountability elements. We don't deal with multi-party *service contract* model in this paper, mainly because most contracts in SaaS only involve two parties. Even if multiple parties are involved in the underlying contract, the *service contract* model can always be decomposed into multiple two-party service contracts. Note that a SaaS normally is not an atomic service that only involves one interaction between a service consumer and a service provider. Instead, it is a composite service that may involve a series of conversations between a service consumer and a service provider. Each conversation is an instance of service contract execution that involves a series of actions performed by both parties. Also during the valid contact period, the service can be executed multiple times, i.e. multiple conversations.

Using the Congo Book service [11] as an example, the *FullCongoBuy* composite service can be offered as a SaaS or Cloud service. The consumer can invoke *FullCongoBuy* many times to buy different books during the valid period of the underlying contract. Each execution may involve a series of actions performed by either the consumer or the provider. An example of a consumer's action can be inputting the book name, whereas a provider's action can be executing the atomic *LocateBook* service for locating the book. Considering this characteristic, Definition 1 does not capture detailed information in each execution. Thus we need another definition for service contract execution:

Definition 2: A service contract execution is a tuple $SCE = (sc, I, Op, Oc, se, R)$, where: *sc* is an individual of service contract *SC*; *I* is execution information, $I = (start\_time, complete\_time, timeout\_value)$; *Op* is a set of obligations (see Definition 1) that are successfully completed by the provider; same applied to *Oc* as the completed obligations by the consumer; *se* is Contract Execution State: $se \in SE$, $SE = \{se_1, se_2, \ldots, se_n\}$, where se_i is one of the user defined contract execution states, for example, *in progress*, *complete*, *pending provider obligation* etc; Rules: $R = \{r_1, r_2, \ldots, r_n\}$, r_j is a horn clause: *consequent* ← *antecedent*. ■

C. Service Contract Model Representation

The above section provides definitions for the structure of our *service contract* model. However, to satisfy requirements listed in Section III.A, we need to first have a representation mechanism. As suggested in our studies in Section II, most of the existing e-Contract models in literature use some logic models with strong expressiveness power to represent a legal contract at the expense of computation decidability. Moreover most of the e-Contract models are theory-based, lack of tooling support and implementation. Conversely, our approach to *service contract* model is to make trade-offs amongst expressiveness, decidability and existing tooling support. The ultimate choice should satisfy the requirements in Section III.A, yet retain computation decidability and can be implemented with existing products and technologies.

As services may vary in different domains, a *service contract* model needs to capture the domain concepts and their relationships, and has the reasoning capability to ensure consistency. An ontology provides exactly these required capabilities; thus a *service contract* model should be based an ontology. In the meantime, it should allow capturing of *service contract* execution information in a knowledge base (KB) so that the *service contract* execution can be tracked and execution states can be reasoned.

An ontology can be specified using Web Ontology Language (OWL), which is recommended by W3C as the standard for representing ontologies on the Web. As a revision of DAML+OIL, OWL provides three sub-languages with increasing level of expressiveness: OWL-Lite (corresponding to $\mathcal{SHIF}(\mathcal{D})$ [12]); OWL-DL (corresponding to $\mathcal{SHOIN}(\mathcal{D})$ [12]); and OWL-Full which is an extension to Resource Definition Framework (RDF). Both OWL-Lite and OWL-DL provide computation completeness and decidability [13], whereas OWL-Full has maximum expressiveness but no computational guaranteed.

We have chosen OWL-DL to represent our *service contract* model since it has the better trade-off between expressiveness and decidability, and it also has mature tooling support. But OWL-DL has its limitations. OWL 1 does not support role chaining. For example, given *hasParent* and *hasBrother* roles, OWL 1 ontology can not entail "*hasUncle*" role. OWL 2 partially solves this through property chains. To address this limitation, we augment OWL-DL with Semantic Web Rule Language (SWRL). SWRL is a W3C submission, extending OWL-DL axioms with a set of horn clause rules. It is basically a combination of OWL-DL/OWL-Lite with the unary/binary Datalog sublanguages of the Rule Markup Language (RuleML) [14]. Therefore, in our case, OWL-DL can be used to define the concepts and roles in our *service contract* ontology while SWRL can be used to define rules for contract execution state reasoning. With this in mind, we here define:

Definition 3. A *service contract* model SC can be represented as a KB that is a triple $Ksc = (\mathcal{T}, \mathcal{A}, \mathcal{H})$, where:

- A TBox $\mathcal{T}$ consists of a finite set of concept inclusion axioms of the form $C \sqsubseteq D$, a finite set of role inclusion axioms of the form $R \sqsubseteq S$ and transitivity axioms $Trans(R)$, where C and D are concepts, R and S are roles;
- An ABox $\mathcal{A}$ consists of a finite set of concept and role assertions and individual equalities/inequalities $C(a)$, $R(a, b)$, $a = b$, and $a \neq b$, respectively;
- A horn rule set $\mathcal{H}$ consists of a finite set of horn clause axioms. A horn rule axiom consists of an antecedent (body) and a consequent (head) in the form of: $a \leftarrow a_1 \wedge a_2 \wedge \ldots \wedge a_n$, where a, a_i $(0 \leqslant i \leqslant n)$ are atoms in rules that can be of the form $C(x)$, $P(x, y)$, $Q(x, z)$, $sameAs(x, y)$ or $differentFrom(x, y)$, and C is a concept; P is an *individual-valued* property; Q is a *data-valued* property; x, y are either variables or individuals; and z is either a variable or a data value. Variables x, y, z must be bound to named individuals in the KB to satisfy the DL-Safe criteria. ■

The other challenge that DL has is that it lacks of an action semantic to describe the dynamic world. In [15], Baader and *et al* integrate action theory into DL $\mathcal{ALCQIO}(\mathcal{D})$

and explore the computation properties of such extension. Based on their approach, the authors in [9] propose a *service contract* model based on $\mathcal{ALCQO}(\mathcal{Q}^*)$, with a slightly different action semantics. While executability and projection are the major concerns in [15], the key concern in our model is determining if an obligation is fulfilled.

We thus introduce an evidence concept and use SWRL rules to simplify action state reasoning. Intuitively, the evidence concept reflects how a particular action's fulfilment is verified in real life. An evidence object, created as a result of the action can be used as a record to prove the occurrence of that particular action. For example in a real life scenario, a receipt can be used as an evidence object to prove that a book selling action has occurred. We adopt a similar action structure as [15, 16] with simplified semantics:

Definition 4: An action is a quadruple $AC = (in, pre, out, post)$ where: *in* is the input of the action, which is a finite set of individuals in *Ksc*; *pre* (precondition) is a finite set of assertions in $\mathcal{A}$, *out* is the action output, which is a finite set of individuals in *Ksc*; and *post* (post-condition) consists of a set of finite set of conditional expressions in the form of φ / χ, where φ is a set of assertions in $\mathcal{A}$, χ is a set of assertions of primitive literals for $\mathcal{T}$. ■

For example the *LocateBook* action: *in* = a book name individual: "*Twin Cities*", *pre* = $\top(a)$, *out* $\in$ {*ISBN*, "*OutofStock*", "*NotFound*"}, *post* = {$\exists$*inStock.Book*(*a*) / *LocatedBook*(*a*), $\neg\exists$*Exists.Book*(*a*) / *NotFoundBook*(*a*), $\exists$*Exists.Book*(*a*) $\sqcap$ $\neg\exists$*inStock.Book*(*a*) / *OutofStock*(*a*) } where *a* is an individual of book "*Twin Cities*".

Definition 5: An evidence object is a triple $E = (obj, timestamp, cond)$ where: *obj* is an individual in *Ksc*; *timestamp* is a data property of *obj* representing the timestamp that *obj* is created; and *cond* is a set of assertions w.r.t. *obj*. ■

An example of evidence can be: *obj* = an acknowledgement message *ack*, *timestamp* = *Timestamp*(*ack*); *cond* = $\exists$*Log.Msg*(*ack*) $\sqcap$ $\exists$*Header.Label*("*Ack_Locate Book*") $\sqcap$ $\exists$*ValidSignature.Msg*(*ack*).

As we use the evidence object as a proxy for the occurrence of an action, we thus have the obligation fulfilment axiom:

Axiom 1: *Obligation* (?*a*) $\wedge$ *mustDo*(?*a*, ?*b*) $\wedge$ *verifiedBy*(?*b*, ?*c*) $\rightarrow$ *fullfilled* (?*a*)

This axiom semantic is equivalent to a trigger rule [17] semantic $C \Rightarrow D$ where C, D are concepts. The trigger rule can be translated into the inclusion axiom with epistemic operator **K** [24]: $\mathbf{K}C \sqsubseteq D$. Intuitively, the **K** operator denotes that the rule only applies to those individuals that KB "knows" to be the instance of concept C, not to arbitrary domain elements. In our case, the rule only applies to those known instances of evidence objects.

In our *service contract* model, a predefined list of valid sequence of actions will regulate the action performing order from both the provider and the consumer. We can use the *List* class as in [18] to represent the action sequence. However, we don't include other control constructs such as *if-else* or *split*, as the actions in *service contract* is more coarse-grained than atomic process in [18]. From a business perspective, the main concern is on the correct performing sequence of the high-level obligations, not on the low level logic of atomic tasks as dealing with traditional workflow.

Based on the above definitions, a *service contract* ontology can be defined. Fig. 1 shows a simplified version of the ontology. The highlighted *scContract* class has contract term definitions (*Defintions* class), and it links to a predefined service *p1:Service* which can be semantically described by OWL-S[18]. The contract class involves *ServiceProvider* and *ServiceConsumer*, both of which have a super class *Party*. Each party has *Obligation* which consists of multiple *Action* and *Evidence* pairs. *scContract* also has a *ActionSequence* class, which defines the valid action sequences. An execution instance of *scContract* will be defined by the highlighted *scContractExecution* class. The *scContractExecution* keeps track of the fulfilled obligations from both the service provider and the service consumer.

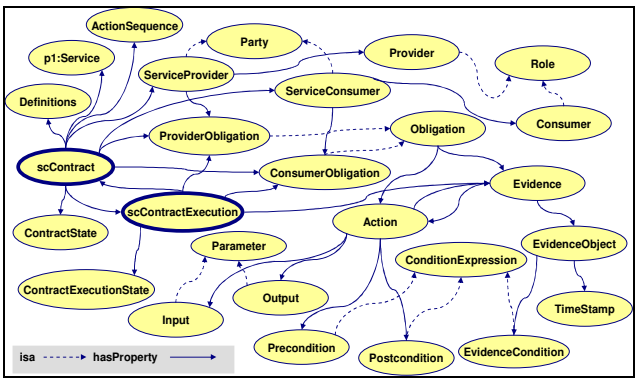

Figure 1. A Simplified Version of the *Service Contract* Ontology

We name a *service contract* model built on top of this ontology as OWL-SC, which complements a service defined by OWL-S with clear obligations spelled out for the service participants. An example is that for *FullCongoBuy* service [18], a domain specific contract class *CongoBookContract* can inherit from the generic *scContract* class. Such domain specific contract instance may be executed multiple times. Each execution can be an instance of *CongBookContractExecution* class, which inherits from the *scContractExecution* class. A KB can monitor the obligation fulfillment situation and moreover, reason the contract state and execution state based on the defined rules in the KB.

D. Properties of Service Contract Model OWL-SC

1) Expressiveness

The underlying DL in OWL-SC is currently $\mathcal{SHOIN}(\mathcal{D})$, with intention to move to OWL 2 $\mathcal{SROIQ}(\mathcal{D})$ when tooling support for OWL 2 is mature. $\mathcal{SHOIN}(\mathcal{D})$'s expressiveness is constrained by its syntax and semantics, which is listed in Fig. 1 in [12]. As mentioned earlier, OWL-DL has limit in role chaining expressiveness. Moreover built-in data type in

OWL 1 is limited to *xsd:string* and *xsd:integer*. In OWL-SC, we augment OWL-DL with SWRL, which extends OWL DL's expressiveness power at two fronts: firstly it allows reasoning of role chaining; secondly, SWRL built-ins can increase expressiveness significantly on *datatypes* and the operations on them. These extensions allow us to bring in reasoning power in action semantics in our model. SWRL's limitation is that it does not allow disjunction and negation in the rules; moreover explicit qualification over rules is also not supported. However, a combined OWL-DL and SWRL can leverage both strengths and provide the expressiveness to satisfy the requirements in section III.A.

2) Computational Properties

While OWL-DL is a decidable logic, SWRL is proven not decidable [19]. As the authors suggest, this is because that DL algorithm can always reach a finite tree model for satisfiability check, but adding the rules breaks the tree model and therefore becomes undecidable. To avoid the problem, the authors propose so called DL-Safe rule. A rule *r* is called DL-safe if each variable in *r* occurs in a non-DL-atom in the rule body. A program *P* is DL-safe if all its rules are DL-Safe (see [19] for more details). The DL-Safe restriction is only exposed to ensure that the variables in the rule body are bound to only explicitly existing individuals in the KB. In our model, anonymous individuals are disregarded as we apply the DL-Safe rule restriction.

From a computational complexity perspective, reasoning in $\mathcal{SHOIN}(\mathcal{D})$ has a worst-case nondeterministic exponential time (NExpTime) [20]. Research on sound and complete reasoning algorithm for OWL DL and rules is still an ongoing effort. Various approaches [21, 22, 23] have been proposed but each has limitations. In particular, we are interested in [22, 23] for our *service contract* model as the reasoning is based on an efficient production rule algorithm – Rete. According to Forgy, Rete's worst complexity for the set of satisfied rules is linear in the number of rules, and polynomial in the number of objects [24].

3) Tooling Support

The basic requirements for tooling in our *service contract* model are the reasoning engine, ontology editor and rule editor. We choose Protégé 3.4.3 [25] as our ontology editor and rule editor. Protégé 3.4.3 supports OWL-DL, it bundles with Pellet and also provides a DIG interface for other reasoners like KAON2 and RACER. Moreover, it bundles with SWRLtab, which allows SWRL rules editing. There is also a SWRLJessTab [22] plug-in available for Protégé 3.4.3, which can translate OWL facts to Jess facts and SWRL rules to Jess Rules. It then allows invocation of the Jess rule engine that implements the RETE algorithm to do reasoning on the translated rules and facts.

4) Meeting the Requirements in Section III.A

Compared to other e-Contract approaches, the differentiation of our *service contract* model is that it is not just a theoretical model, but can be practically implemented with existing standards and tools; moreover, the action semantics and evidence concept closely mimic how accountability is treated in real situations in business domain. In summary, with OWL-SC, obligations can be clearly specified, and interpreted by software agents. This satisfies AR 1 in Section III.A. Secondly, obligations specified in OWL-SC can also be referred through URI on the web, which meets AR 2 and suits the SaaS and Cloud environment. Finally the action semantic and the concept of evidence of OWL-SC allow accountability solutions to be built to satisfy AR 3 and AR 4 respectively.

IV. A GRAPHICAL MODEL– SC-CPN

A. Coloured Petri-nets

OWL-SC provides a structure and semantics for modelling obligations and actions in a service contract. But neither OWL-DL nor Protégé provides an easy way for modelling the sequence of the obligations. On the other hand, Coloured Petri-Nets (CP-Nets) provide an intuitive graphical representation underpinned by a rigorous mathematics foundation; and more importantly, CP-Nets have an explicit semantic to describe both actions and states whereas most other formalisms can only focus on one aspect. Furthermore, tools like CPN-Tools [26] are available to assist in net editing, simulation and state space analysis. Thus CP-Nets are ideal for visual modelling, analysis and validation of our *service contract* model. We first outline the definition of CP-Nets from [27]:

Definition 6: A CP-Net is a tuple $CPN = (\Sigma, P, T, A, N, C, G, E, I)$ where: Σ is a finite set of non-empty types, also called colour sets; P is a finite set of places; T is a finite set of transitions; A is a finite set of arcs such that: $P \cap T = P \cap A = T \cap A = \emptyset$; N is a node function. It is defined from A into $P \times T \cup T \times P$; C is a colour function. It is defined from P into Σ; G is a guard function. It is defined from T into expressions such that: $\forall t \in T$: $[Type(G(t)) = \mathbf{B} \wedge Type(Var(G(t))) \subseteq \Sigma]$; E is an arc expression function. It is defined from A into expressions such that: $\forall a \in A$: $[Type(E(a)) = C(p)_{MS} \wedge Type(Var(E(a))) \subseteq \Sigma]$ where p is the place of $N(a)$; I is an **initialisation** function. It is defined from P into closed expressions, such that: $\forall p \in P$: $[Type(I(p)) = C(p)_{MS}]$. ■

B. CP-Nets Model for Service Contract – SC-CPN

We now define a set of one-to-one mapping rules to map action semantics in OWL-SC to a CP-Net.

Definition 7: Given a KB defined in OWL-SC $Ksc = (\mathcal{T}, \mathcal{A}, \mathcal{H})$, we map each concept in $\mathcal{T}$ to a colour set ∂ in a place p, $\partial \in \Sigma, p \in P$; each assertion in $\mathcal{A}$ to a constant or variable declaration in the CP-Net, therefore each model of $\mathcal{T}$ and $\mathcal{A}$ in $\mathcal{J}$ corresponds to an initial marking in M_0. We also map each consumer action in Ac to a consumer transition in Tc, each provider action in Ap to a provider transition in Tp, where $Tc \cap Tp = \emptyset$, $Tc \cup Tp = T$; input *in* and output *out* to token colours in Pi, where $Pi = {}^\bullet Tp \cap Tc^\bullet \cup {}^\bullet Tc \cap Tp^\bullet$, $Pi \subseteq P$; precondition *pre* to $i \wedge g$, where i is a set of token colours in ${}^\bullet T$ that satisfies with input inscription Ei, $Ei \subseteq E$, $g \in G$, g is a transition guard in T; also map post-condition *post* to Eo , where Eo is output arc inscription, $Eo \subseteq E$, $Ei \cap Eo = \emptyset$. ■

Based on Definition 7, we can construct a CP-Net from OWL-SC, we call it as *service contract* CP-Net— *SC-CPN*.

Thus the standard CP-Net analysis techniques can be applied to validate the correctness of the *service contract* model w.r.t. the consumer and provider behaviour. Intuitively, we can obtain two theorems on executability and projection (see [15] for their definitions) based on the mapping rules in Definition 7.

Theorem 1: Action executability problem in OWL-SC can be translated to a transition fireability problem in SC-CPN.

***Proof Sketch*:** Based on the mapping rules, it is obvious that the theorem holds for one action. Assume it holds for $(a_1, ..., a_k)$ where $1 \leqslant i < k$, i.e., for transitions $(t_1, \ldots, t_k)$, if $t_1..t_k$ are fireable, then for all models $\mathcal{J}$ of $\mathcal{T}$ and $\mathcal{A}$, all interpretations $\mathcal{J}'$ with $\mathcal{J} \Rightarrow \mathcal{T}a_1, \ldots, a_i\,\mathcal{J}'$, we have $\mathcal{J}' \models pre_{i+1}$. Now assume transition t_{k+1} in $(t_1, \ldots, t_{k+1})$ sequence is also fireable. As defined in the mapping rules, all initial markings M_0 correspond to all models $\mathcal{J}$ of $\mathcal{T}$ and $\mathcal{A}$. Prior to transition t_{k+1}, the marking is m_k, which is fireable, i.e. tokens at ${}^\bullet t_{k+1}$ satisfy input inscription ei, as well as guard g at t_{k+1}, this implies that pre_{k+1} holds. Since m_k corresponds to $\mathcal{J}''$, *where* $\mathcal{J}' \Rightarrow \mathcal{T}a_i, ..., a_j\,\mathcal{J}''$, $i \leqslant j < k+1$, $\mathcal{J}'' \models pre_{j+1}$, and therefore, $\mathcal{J} \Rightarrow \mathcal{T}a_1, .., a_i, \ldots, a_j\,\mathcal{J}''$, $i \leqslant j < k+1$, which suggests $(a_1, \ldots, a_{k+1})$ is executable.

Theorem 2: Action projection problem in OWL-SC can be translated to reachability problem in SC-CPN.

Similar approach like the one in Theorem 1 can be applied to prove this theorem. We omit it due to space limit.

The *ActionSequence* in *scContract* contains the valid action sequences to regulate the consumer and provider interactions. Therefore to check whether or not a service contract is breached in an execution instance, it is just a matter of simply checking the actual action sequence in *scContractExecution*, to see if it matches to any predefined action sequence in *ActionSequence* in *scContract*. If not matched, by locating which party's action causes the discrepancy we can identify the responsible party.

C. SC-CPN Properties

Due to the space limit, we briefly list the desired properties of SC-CPN w.r.t. our *service contract* model.

Structural Boundedness: Assume place p in $\langle \mathcal{N}, m_0 \rangle$, $\mathbf{b}(p) = \sup\{m[p] \mid m \in \mathrm{RS}(\mathcal{N}, m_0)\}$, a net is structural boundedness iff $\forall p\ \mathbf{b}(p) < \infty$, where sup is the token upper bound of p, RS is a reachability set.

Liveness Properties: We always expect that the net can be terminated at some state, i.e. there exists at least one dead marking for any initial marking of the net. Also, each transaction should have the possibility of firing, i.e. no dead transition exists. The SC-CPN liveness property can be described as: $\exists m \in \mathrm{RS}(\mathcal{N}, m_0)$, for $\forall t \in T$ such that t is not fireable at m, and for $\forall t \in T$, $\exists m \in \mathrm{RS}(\mathcal{N}, m_0)$, $\exists \sigma$ such that $m \xrightarrow{\sigma t} m'$.

Reversibility: We expect that the execution of service will lead to a new state. Thus non-reversibility is a preferred property for SC-CPN, which is defined as: $\exists m \in \mathrm{RS}(\mathcal{N}, m_0)$, $\nexists \sigma$ such that $m \xrightarrow{\sigma} m_0$.

D. Using SC-CPN to Model Service Contract Behaviour

SC-CPN provides an intuitive graphical model for modelling action sequence for OWL-SC. Simulation can be used to identify errors in the service contract's action model. Standard Petri-nets state and reachability analysis can be used to analyse the executability and projection in OWL-SC.

V. Using OWL-SC & SC-CPN to Model Service Contract for Congo Book Service

Now we illustrate how to use OWL-SC and SC-CPN to build a *service contract* model for the Congo Book service, which is a widely used example of semantic web service. We will first define the basic concepts and SWRL rules in OWL-SC; then we use SC-CPN to model the obligation and action sequence; next we will use Pallet reasoner to classify and check consistency of the ontology; then we will use Protégé's SWRLJessTab to translate the OWL individuals to Jess facts, SWRL rules to Jess rules and use Jess to entail new facts; and finally we will use SQWRLQuerytab to query the *service contract* knowledge base.

A. Defining Contract Service Ontology for CongoBook

We now use Protégé 3.4.3 to create the ontology based on the model in Fig. 1. Firstly we extent *scContract* class to create *CongoBookServiceContract* as the *service contract* class for Congo Book service, then subclass *scContractExecution* to create *CongoBookServiceExecution* for the contract execution class. After that we can enter some individuals. We use some candidate actions based on the Congo Book semantic service *CongoProcess.owl*[18] as a starting point: *P_LocateBook*, *P_SignInUser*, *P_PutInCart*, etc.

B. Modeling Obligations using SC-CPN

Now we use SC-CPN to model the actions and verify the correct sequence of actions. We leverage CPN-Tools' hierarchical net feature [34] to create a hierarchy of *service contract* nets across multiple pages for ease of modeling. Fig. 2 shows the hierarchy page. Fig. 3 shows the top page of *CongoBookServiceContract*, which can be used as a generic high-level SC-CPN model for SaaS or Cloud services. In the model, *ServiceConsumer* interacts with *ServiceProvider* through the *ServiceInput* and *ServiceOutput* places; each party maintains session information such as shopping cart, credentials in *ServiceSession* place; and *ServiceConsumer* pays for service with consideration and *ServiceProvider* delivers service effect.

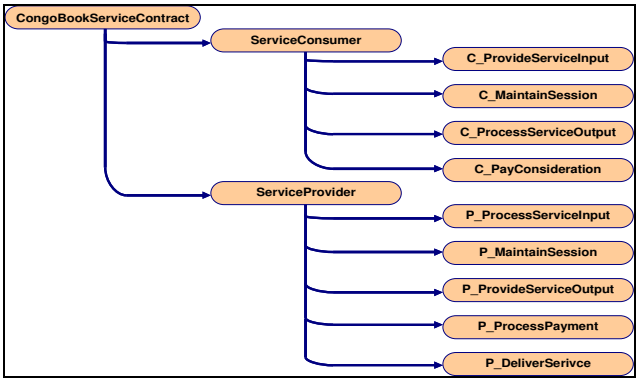

Figure 2. *CongoBookServiceContract* Hierarchy Page

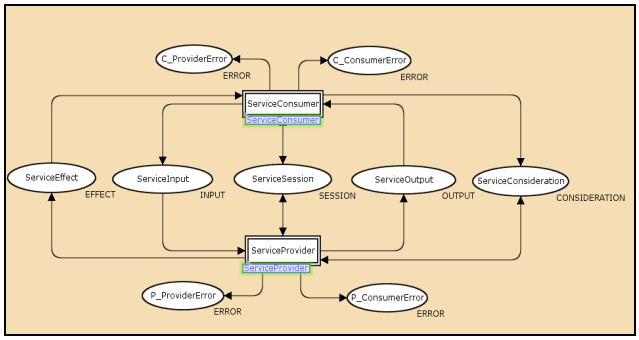

Figure 3. *CongoBookServiceContract* Top Page

Fig. 4 shows the *ServiceConsumer* page, which further breaks down the transitions to another level. Fig. 5 shows the *DeliverService* page, which models the lowest granularity of a transition, in this case is the Service Provider's obligation of Shipping book to the consumer.

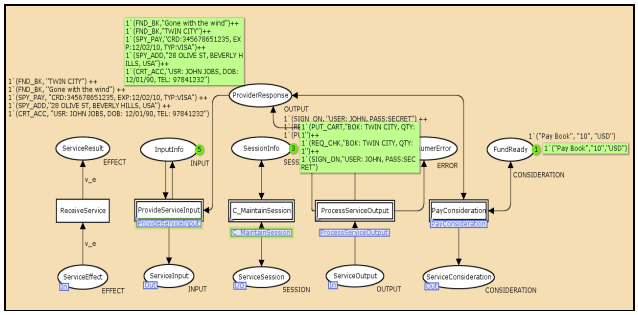

Figure 4. *ServiceConsumer* Page

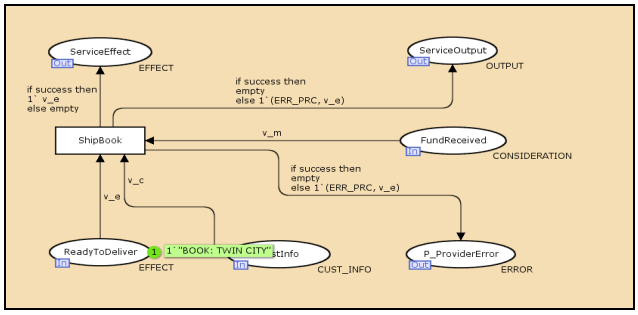

Figure 5. *DeliverService* Page

Once all of the net pages have been constructed, we run simulations to see the inter-play between service consumer and service provider. We next use CPN-Tools' state space analysis capability to analyze the SC-CPN properties. We can continue to refine the SC-CPN model based on the desired properties discussed in Section IV.C. Then from the reachability graph, we produce the valid action sequence for OWL-SC. For example, one normal action sequence is: *C_ProvBookName*, *P_LocateBook*, *C_PutInCart*, *P_GetSignOn*, *C_ProvSignOn*, *P_SignUserIn*, *P_PutInCart*, *C_Checkout*, *P_GetPaymentInfo*, *C_ProvPaymentInfo*, *P_ProcPayment*, *P_GetAddInfo*, *C_ProvAddInfo*, *P_ReqAuthPay*, *C_AuthorizePay*, *P_ShipBook*, *C_AcceptBook*. Another sequence could be just *C_ProvideBookName*, *P_LocateBook*, where the consumer either just wants to search without intention to pursue the purchase; or the book may be out of stock. Other valid action sequences are omitted due to the page limit.

C. *Reasoning of Congo Book Service Contract Model*

With the valid action sequences identified, we now add the action sequence instances into the ontology. Then we input some evidence instances to simulate the collection of action evidences in the *service contract* KB. Finally we create rules to reason the completion of obligations and the states of the contract execution. For example, axiom 1 for *CongoBookServiceContract* can be input in SWRLTab as:

$$ServiceContractExecution(?x) \wedge execute(?x, ?y) \wedge specifiedObligation(?y, ?z) \wedge mustDo(?z, ?a) \wedge verifiedBy(?a, ?b) \wedge produceEvidence(?x, ?b) \rightarrow fulfilledObligations(?x,?z)$$

Once the ontology and rules are finally built, we invoke Pellet reasoner to classify the TBox and check consistency of the ABox. Then through the SWRLJessTab, we convert OWL individuals/SWRL rules to Jess facts/rules respectively; and use Jess engines to do reasoning which can entail new Jess facts. Fig. 6 shows an example of the reasoning result. As we can see, the Jess engine inferred a lot of facts about fulfilled obligations based on evidence produced by contract execution instance *CongoBookServiceExecution_1*. It also concluded that the *CongoBookServiceExecution_1* is in completion status based on the highlighted rule. After reasoning, we can use SPARQL or SQWRLQueryTab to query the KB.

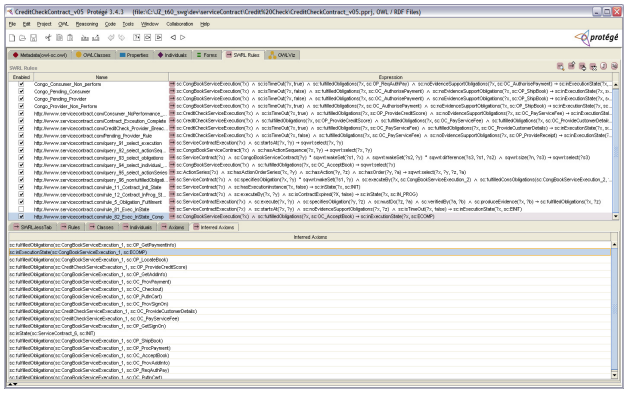

Figure 6 Congo Book Service Contract Reasoning Results

D. *Discussions*

So far we have demonstrated the modelling of a pro-accountability *service contract* model for an online service using existing tools. Clearly this *service contract* model can be applied to other SaaS and Cloud services to support service obligation disclosure, monitoring and tracking of non-compliance. The strengths of our model are that firstly it addresses the accountability concerns through a formal construct, which is firmly grounded on the intuitive concept of service contract in the real world. Secondly the construct is based on rigorous formalisms like OWL-DL/DL-Safe SWRL and CP-Nets. The former allows machine interpretation and web accessibility, yet provides computation decidability. The later facilitates visual modeling and simulation, yet is backed by solid mathematics.

Lastly our approach can be supported by existing tools and readily applied in practical applications, rather than just a theoretical model. We also notice that the weakness of the model is in its expressiveness and reasoning power. For example, it is quite difficult to reason whether a contract is expired, since SWRL's Temporal built-in [28] currently does not provide the support for current time (e.g. now). Another issue is that the translation from OWL-DL to Jess is not complete; the anonymous individuals will not be translated to Jess facts [23]. However this limitation does not impact the completeness of our model because anonymous individual information is not relevant to our model. The last issue is that the inferred facts from Jess reasoning can potentially make the initial OWL-DL ontology inconsistent [23]. We suggest that a hybrid approach that leverages strengths from different formalisms can best address the weakness. We separate our service contract reasoning to design-time reasoning and run-time reasoning stages. We only use DL reasoner for concept consistency checking at design-time while Jess rule engine is used to maintain the KB at run-time. Other tasks such as comparison and computation can be better handled via a combination of traditional programming model and KB query. For example, checking whether or not a contract is expired, it is easy to query the KB, get the end time; then compare it to the current time in a traditional program.

VI. Conclusion and Future Work

High accountability standard not only benefits service consumers as a whole, but also can be a differentiator for service provider. As such, it is paramount to enable accountability in SaaS and Cloud services. In contrast with existing approaches that address accountability issues at a technical protocol level, we address them at an architecture level through a pro-accountability service contract formalism, which closely mimics the contract concept in the commercial world.

Our contribution can be therefore summarized as: firstly, we analyze the accountability management requirements for SaaS and Cloud services and define a formal construct for a pro-accountability *service contract* model, proposing the unique concepts such as *service contract execution* and action evidence that are not seen in other e-Contract models. We adopt the decidable OWL-DL, coupling with the enhanced action semantics and DL-Safe SWRL rules to represent the *service contract* construct, namely OWL-SC. We also propose a novel approach to map OWL-SC action semantics to a Colored Petri-Nets model, namely SC-CPN, and thus enable visual modeling, validation and simulation of action model in OWL-SC. We have used the Congo Book service as an example to demonstrate how to use existing tools to build the *service contract* model, and discussed the strengths and weaknesses of the current model plus the recommended approach to address the weaknesses. Other domain specific application of such model is a future work.

References

[1] A. Schedler, Self-Restraining State: Power and Accountability in New Democracies, Lynne Reiner Publishers, 1999, pp. 13-28.

[2] Kwei-Jay Lin, Joe Zou and Yan Wang, Key Note, Accountability Computing for e-Society, The International Conference on Advanced Information Networking and Applications, IEEE, 2010.

[3] A. Dan and et al.: Business-to-Business Integration with TPAML and a Business-to-Business Protocol Framework, IBM System Journal, 40(1), IBM, 2001.

[4] L. Xu, Monitoring Multi-party Contracts for E-business, Dissertation, Doctor of Philosophy, University of Toronto, 2004.

[5] A. Daskalopulu, Logic-Based Tools for the Analysis and Representation of Legal Contracts, Dissertation, Doctor of Philosophy, University of London, 1999.

[6] G. Governatori and Z. Milosevic, A Formal Analysis of a Business Contract, Language, Proc. Int'l J. Cooperative Info. Sys., vol. 15, no. 4, 2006, pp. 659–685.

[7] A.D.H. Farrell and et al, Performance Monitoring of Service-Level Agreements for Utility Computing Using the Event Calculus, Proc. 1st Int'l W'shop Electronic Contracting, IEEE CS Press, 2004, pp. 17–24.

[8] H. Davulcu, M. Kifer, and I.V. Ramakrishnan, "Ctr-s: A logic for specifying contracts in semantic web services," in Proc. WWW 2004, ACM, May 2004, pp. 144–153.

[9] H. Liu and et al. Modeling and Reasoning about Semantic Web Services Contract using Description Logic, The Ninth International Conference on Web-Age Information Management, IEEE, 2008.

[10] B. Grosof and T. Poon, SweetDeal: Representing Agent Contracts with Exceptions Using XML Rules, Proc. 12th Int'l Conf. World Wide Web, ACM Press, 2003, pp. 340–349.

[11] A. Ankolekar and et al, DAML-S: Web Service Description for the Semantic Web, DAML-S Coalition, Proc. Int'l Semantic Web Conf. (ISWC), LNCS 2342, Springer Verlag, 2002, pp. 348-363.

[12] I. Horrocks, P.F. Patel-Schneider and F.V. Harmelen, From SHIQ and RDF to OWL: The making of a web ontology language. J. of Web Semantics, 2003, 1(1):7–26.

[13] W3C, OWL web ontology language overview. http://www.w3.org/TR/owl-features/

[14] I. Horrocks and P.F. Patel-Schneider, A proposal for an OWL rules language. In The Thirteenth International World Wide Web Conference, New York, May ACM Press, 2004.

[15] F. Baader and et al, Integrating description logics and action formalisms for reasoning about web services, LTCS-Report 05-02, 2005. http://lat.inf.tudresden.de/research/reports.html.

[16] S. Narayanan and S. A. McIlraith, Simulation, Verification and Automated Composition of Web Services, 11th International World Wide Web Conference, 2002.

[17] F. Baader, D. Calvanese and et al, The Description Logic Handbook, Theory, Implementation and Applications, Cambridge Uni., 2003.

[18] D. Martin and et al, Describing Web Services using OWL-S and WSDL, available at: http://www.ai.sri.com/daml/services/owl-s/1.1/owl-s-wsdl.html

[19] B. Motik, U. Sattler and R, Studer, Query Answering for OWLDL with Rules, The SemanticWeb ISWC: 3rd International Semantic Web Conference, Hiroshima, Japan, 2004.

[20] I. Horrocks and P.F. Patel-Schneider, Reducing OWL Entailment to Description Logic Satisability, 2nd Intl. Semantic Web Conference, Florida, USA, Oct, 2003

[21] A. Y. Levy and M.C. Rousset, Combining Horn rules and description logics in CARIN, Artificial Intelligence 104 (1998) 165-209, 1998.

[22] C. Golbreich and A. Imai, Combining SWRL rules and OWL Ontologies with Protégé OWL Plugin, Jess, and Racer. 7th International Protégé Conference, Bethesda, MD, 2004.

[23] J. Mei, and E. Paslaru Bontas, Reasoning Paradigms for SWRL-enabled Ontologies,Protégé With Rules Workshop, Madrid, 2005.

[24] C.L. Forgy, On the Efficient Implementation of Production Systems, Ph.D. dissertation, Carnegie-Mellon University, 1979.

[25] Protégé, Available at http://protege.stanford.edu/

[26] CPN-Tools, http://wiki.daimi.au.dk/cpntools-help/_home.wiki

[27] K. Jensen: An Introduction to the Theoretical Aspects of Coloured Petri Nets. In: J.W. de Bakker, W.-P. de Roever, G. Rozenberg (eds.): A Decade of Concurrency, Lecture Notes in Computer Science vol. 803, Springer-Verlag 1994, 230-272.

[28] Protegewiki, SWRLTemporalBuiltIns, 2002, Available at http://protege.cim3.net/cgi-bin/wiki.pl?SWRLTemporalBuiltIns.