Modeling Service Level Agreements with Binary Decision Diagrams

Constantinos Kotsokalis, Ramin Yahyapour, and Miguel Angel Rojas Gonzalez

Dortmund University of Technology, Germany
constantinos.kotsokalis@udo.edu, ramin.yahyapour@udo.edu,
miguel.rojas@udo.edu

Abstract. The vision of automated service composition for enabling service economies is challenged by many theoretical and technical limitations of current technologies. There is a need for complete, dependable service hierarchies created on-the-fly for critical business environments. Such automatically-constructed, complex and dynamic service hierarchies imply a similarly automated process for establishing the contracts that specify the rules governing the consumption of services; and for binding them into respective contract hierarchies. Deducing these required contracts is a computationally challenging task. This also applies to the optimization of such contract sets to maximize utility. We propose the application of (Shared) Reduced Ordered Binary Decision Diagrams, a suitable graph-based data structure well-known in the area of Electronic Design Automation. These diagrams can be used as a canonical representation of SLAs, thus allowing their efficient and unambiguous management independent of their structure's specifics. As such, this representation can facilitate the process of negotiating SLAs, subcontracting parts of them, optimizing their utility, and managing them during runtime for monitoring and enforcement.[1]

1 Introduction

Recent trends in service computing are lead by the vision of an *Internet of Services*, a marketplace without boundaries where service economies can flourish through composition and re-use. Suitable mechanisms, and the automation achieved through smart agents, will be the key enabler for this goal. It is anticipated that, eventually, full potential can be achieved through the automation of contracting for such services. More specifically, it is desired that service consumption can be enabled with determinism, under well-specified contracts that define all parameters and govern the use of a service by its customer.

Such a contract is encoded in a *Service Level Agreement* (SLA). A SLA is essentially a set of *facts*, and a set of *rules*. Facts are globally (with respect to the contract) applicable truths, such as parties involved, monetary unit, etc. Rules include:

[1] The research leading to these results is supported by the European Community's Seventh Framework Programme (FP7/2007-2013) and the SLA@SOI project under grant agreement no.216556.

L. Baresi, C.-H. Chi, and J. Suzuki (Eds.): ICSOC-ServiceWave 2009, LNCS 5900, pp. 190–204, 2009.

1. the *conditions* that must hold for a certain *clause* to be in effect;
2. the clause itself, typically describing the expected result that the customer wishes to receive – and which is usually referred to as *Service Level Objective* (SLO); and
3. a fall-back clause in the case that the aforementioned clause is not honored.

As an example, for the condition "time of day is after 08:00", the clause could be "response time is less than 5 seconds", and the fall-back clause could be an applicable penalty. This kind of format actually reflects real-life contracts and their *if-then-else* structure, which might apply either as the default or as the exception to such default respectively.

In this paper we propose that a *graph-based* data structure, well-known in the domain of *Computer Aided Design* (CAD) for *Very Large Scale Integrated* (VLSI) circuits, is suitable for modeling SLAs in a way which is both expressive enough, and very efficient. *Reduced Ordered Binary Decision Diagrams* (ROBBDs) were introduced by R. Bryant in 1986 [1] as an evolution of C.Y. Lee's [2] and S. Akers' [3] work on BDDs. The hardware industry race has further contributed to the optimization of the structure itself with a significant amount of relevant research, and a large number of methods already exist for taking advantage of ROBDDs' inherent properties.

The essential reason that ROBDDs are useful for modeling SLAs, is that they are canonical representations generated on the grounds of *if-then-else* rules. As such, they can express SLAs unambiguously: equivalent SLAs which are *structurally* different, are eventually represented by the same ROBDD. On the contrary, using formats developed for *on-the-wire* representation such as WS-Agreement [4] or WSLA [5] does not guarantee this property. We propose that ROBDDs are used internally in systems which have to manage SLAs, as a representation that facilitates their management. Suitable interpreters should then be developed to convert from standardized, interchangeable formats such as WS-Agreement and WSLA, to this more convenient data structure and vice-versa.

This paper continues with Section 2, which is discussing related work on SLA representation, management of hierarchies, and previous efforts to relate them to Logic. Following, Section 3 elaborates on (Shared) ROBDDs. Section 4 details their relationship with SLAs and the specific proposal on how to use them for our purposes. Section 5 illustrates initial experimental results. Finally, Section 6 concludes the paper with a summary of core results, and an outlook to future work.

2 Related Work

BDDs are classified as a tool in the area of symbolic model checking. This is the scientific discipline looking into the problem of verifying that a given system satisfies specific requirements, given any kind of input. To our best knowledge, this is the first work that uses BDDs to model and verify SLAs and SLA dependencies. That said, BDDs have been used in service computing before, albeit in very few occasions. In [6] the authors are using a special form of BDDs,

called *Zero-Suppressed* BDDs, to create compact digests of service advertisements. Then, the digests are distributed to interested parties which use them for their service composition needs. In [7], the authors are using BDDs for matching service advertisements in publish-subscribe systems (making use of equivalence checking).

As regards SLA modeling in general, the most well-known efforts are WS-Agreement and WSLA. As also mentioned in the previous section, the focus of these specifications is on-the-wire representations for enabling interoperability between independent agents. This is an area we are not targeting with the work presented in this paper; rather, our focus is a system-internal representation, that will enable efficient mechanisms for decision making.

With regards to applying logic-based approaches to the topic of SLA management, the work which comes closest to ours is the one described in [8]. There, the authors look at the problem in more detail, defining constructs also for things such service description, pricing, QoS, etc. On the other hand, we face everything in an abstract way here, and assume external syntactical definitions and appropriate architectural patterns for applying these definitions. Additional differences include our explicit focus on managing hierarchies of SLAs and associations between them as such. The necessary constructs for this kind of functionality also exist in [8], however there is no mention of essential facilities such as equivalence checks and translation between different vocabularies for different layers of a complete IT stack.

3 Binary Decision Diagrams

This section serves as a general, high-level introduction to BDDs and their basic properties. Motivated readers are encouraged to consult with the bibliography for in-depth material. Most of the definitions provided in this section, are summaries of the definitions that can be found in [9].

A BDD is a graph-based representation of one or more boolean functions. This kind of diagram is based on *Shannon's decomposition theorem* [10], which states that, assuming a boolean function $f : X_n \rightarrow X_m$, where $X_n = \{x_1, ..., x_n\}$ and $X_m = \{x_1, ..., x_m\}$, then for any boolean variable x_i, $i \leq 1 \leq n$:

$$f = x_i \cdot f_{x_i=1} + \overline{x_i} \cdot f_{x_i=0} \tag{1}$$

What Equation 1 provides, is the *if-then-else* representation we are looking for: If x_i is *true*, *then* $f_{x_i=1}$ must be evaluated, or *else* $f_{x_i=0}$ must be evaluated. A BDD then, is a *directed acyclic graph* $G = (V, E)$, where V denotes the vertices (nodes) and E the edges. Vertices can be either *terminal* (i.e. their out-degree is equal to zero), or *non-terminal*. The former can carry a value of either **1** (*true*) or **0** (*false*). The latter are labelled with a variable $x_i \in X_n$; if u is the node, the variable x_i is referred to as $var(u)$. Of the two children nodes, the one followed if x_i evaluates to *true* is referred to as $then(u)$, and the other as $else(u)$.

An illustrative example can be found in [9]. This example is shown in Figure 1(a), where we see a BDD representation of the boolean function $f =$

$x_1 \cdot x_2 + \overline{x_1} \cdot x_3$. We typically use solid lines for the edge between u and $then(u)$, and dashed lines for the edge between u and $else(u)$. Additionally, non-terminal nodes are denoted as circles, while terminal nodes as squares.

Let π be an ordering of the boolean variables involved in the function to represent. Then, the pair (π, G) is the *Ordered BDD* (OBDD) representation of the function, as long as (additionally to simple BDD definitions) it is true that on each path from the root to a terminal node the variables are encountered at most once and in the same order. Looking into the previous example, Figure 1(b) is illustrating exactly this ordering of variables, and how it affects the diagram. A diagram with more than one roots (i.e., representing more than one boolean functions which depend on the same boolean variables) is a *Shared BDD* (SBDD). It must be noted that a root node here does not necessarily imply that the in-degree of this node is equal to zero. For a specific function within a BDD or a SBDD, a *path* is a subset of G which connects the root with a terminal node, without any duplicate occurrences of a node or an edge. We denote the set of all paths for function f as Γ_f.

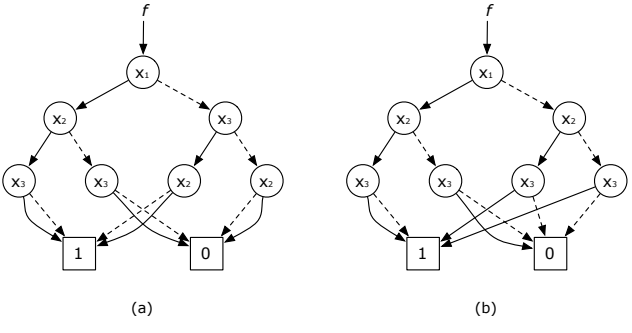

Fig. 1. Simple/Ordered BDD representations of $f = x_1 \cdot x_2 + \overline{x_1} \cdot x_3$

Last before looking at how this kind of diagrams facilitates our work for SLAs, is a short introduction to their operations for reduction. BDDs can be reduced in two ways:

1. *Deletion*: If for a non-terminal node u of G it is true that $then(u) = else(u) = u'$, the node can be removed from the graph. All edges pointing to it, if any, must now point to u', and if u was a root node, then u' must be upgraded to a root node.
2. *Merging*: If for two non-terminal nodes u and u' it is true that $var(u) = var(u')$, $then(u) = then(u')$ and $else(u) = else(u')$, then it is possible to remove u and have all edges pointing to it redirected to point to u'. Additionally, if u is a root node, then u' must be made into a root node.

Remark. In the text that follows, we will use the term *BDD* universally, to refer to Reduced Ordered BDDs. Also, we will not distinguish between single-rooted

and shared diagrams. Whenever single-rooted BDDs are *explicitly excluded*, we will denote that by pre-pending "shared" or just the letter "S".

4 SLAs as BDDs

4.1 A Motivating Scenario

Let us now consider a somewhat typical (albeit reduced, for this example) scenario, where SLA management is necessary. We are assuming an *Infrastructure as a Service* (IaaS) provider; a *Software as a Service* (SaaS) provider which is also a customer to the IaaS provider; and an end-customer of the SaaS provider. We are therefore working on the assumption that the SaaS provider has no infrastructure of its own, therefore all operations are outsourced to the IaaS provider who owns the infrastructure for the software to be executed. This kind of business scenario involves two SLAs, as shown in Figure 2. The first (SLA-1) is established between the end-customer and the SaaS provider, to govern their interactions and apply guarantees. The second (SLA-2) is established between the SaaS and the IaaS providers for the same purpose.

Fig. 2. A scenario with a SaaS and an IaaS provider

The end-customer certainly is not interested in the physical or virtual resources that the software will execute on, in order to receive performance which is acceptable. Therefore, the customer would try to engage in a SLA with the SaaS provider, which would involve –for instance– metrics for service availability, and service invocations completion time (CT). The SaaS provider would typically have some understanding about the software based on modeling principles or historical monitoring evidence, starting from which it can derive expected resource requirements, possibly varying throughout a day's, month's or other period. The infrastructure resource requirements, on the other hand, would be the guarantees that the SaaS provider's SLA with the IaaS provider would need to include. Our example SLAs are described as follows:

SLA-1: For service "Service-1", and given that business hours are between 09:00 and 17:00: During business hours, operation "Operation-1" must complete within 5 seconds, and the service's availability must be more than 99%. Outside business hours, completion time for the same operation can be up to 10 seconds, and the service's availability must be more than 95%.

SLA-2: For service "VMpool", and given that business hours are between 09:00 and 17:00: During business hours, 10 virtual machines must be allocated to this contract. Outside business hours, 5 virtual machines must be allocated.

Table 1. Example clauses

SLA	Variable	Proposition	Proposition type
SLA-1	x_1	ServiceName = 'Service1'	Fact
SLA-1	x_2	BusinessHours = 09:00 - 17:00	Fact
SLA-1	x_3	TimeOfDay in BusinessHours	Condition
SLA-1	x_4	'Operation1' CT $<$ 5 sec	Clause
SLA-1	x_5	Service1 availability $>$ 99%	Clause
SLA-1	$\overline{x_3}$	TimeOfDay not in BusinessHours	Condition
SLA-1	x_6	'Operation1' CT $<$ 10 sec	Clause
SLA-1	x_7	Service1 availability $>$ 95%	Clause
SLA-2	y_1	ServiceName = 'VMpool'	Fact
SLA-2	y_2	BusinessHours = 09:00 - 17:00	Fact
SLA-2	y_3	TimeOfDay in BusinessHours	Condition
SLA-2	y_4	Number of VMs = 10	Clause
SLA-2	$\overline{y_3}$	TimeOfDay not in BusinessHours	Condition
SLA-2	y_5	Number of VMs = 5	Clause

Table 1 illustrates the set of facts and clauses that we will use for this example scenario. It is straightforward to see that, given these facts and clauses in the form of boolean variables which evaluate to *true* or *false*, the SLAs can also evaluate correctly if they are modeled according to Equations 2 and 3 respectively. In the upcoming Section 4.2 we will formalize the problem of expressing SLAs as boolean functions. Then in Section 4.3 we will show how these specific example SLAs map to BDDs.

$$f = x_1 \cdot x_2 \cdot (x_3 \cdot x_4 \cdot x_5 + \overline{x_3} \cdot x_6 \cdot x_7) \tag{2}$$

$$g = y_1 \cdot y_2 \cdot (y_3 \cdot y_4 + \overline{y_3} \cdot y_5) \tag{3}$$

4.2 SLAs and SLA Hierarchies

In Section 1 we referred briefly to service hierarchies and the corresponding SLA hierarchies. Each SLA governs the consumption of one or more services, by one or more consumers. Involved parties have specific obligations to comply with and/or specific gains to expect. In order to carry out its obligations, a service provider involved in a SLA may have to *subcontract*, that is to establish one or more additional SLAs with parties not directly involved in the initial one. This kind of dependency between the original contract and the subcontracts may take many different forms. It may be related to capacity, functionality limitations, failover capabilities, or may represent some other aspect of the provider's *modus operandi* and business model. As such, it is very generic and makes it difficult to identify exactly how the state of one contract affects the state of another.

We formulate a proposed SLA representation as follows: Let Φ^n be the universe of *facts* applicable to contracts as indisputable truth, $\Phi^n = \{\phi_1, ..., \phi_n\}$. Also let

Y^m be the universe of *clauses* which can be evaluated to either *true* or $false$, $Y^m = \{y_1, ..., y_m\}$. A Service Level Agreement is the boolean function f:

$$f : F^k \cup Z^l \rightarrow \{0, 1\} \tag{4}$$

where $F^k \subseteq \Phi^n$, $F^k = \{\phi_1, ..., \phi_k\}$ and $Z^l \subseteq Y^m$, $Z^l = \{z_1, ..., z_l\}$.

We therefore have a representation of a SLA as a boolean function, taking advantage of a SLA's binary nature upon evaluation as *possible / impossible* to satisfy (at negotiation time) or *honored / violated* (at runtime, i.e. while the service is being consumed). The variable terms of a SLA are taking values from Z^l, while pre-agreed understanding and in general facts about the world is encoded in facts accepting values from F^k. This definition is broad enough to encompass various previous definitions, both conceptual (e.g. [11]) and syntactical (e.g. WS-Agreement).

We are now ready to codify SLA dependencies in a generic way, that allows enough flexibility to describe any such kind. Let:

- $f : F^k \cup Z^l \rightarrow \{0, 1\}$, the dependent SLA
- $f_i : F^{ki} \cup Z^{li} \rightarrow \{0, 1\}$, $i \in \mathbb{N}$, the depending SLAs
- $F^{ki} \subseteq \Phi^n$, $F^{ki} = \{\phi_{1i}, ..., \phi_{ki}\}$
- $Z^{li} \subseteq Y^m$, $Z^{li} = \{z_{1i}, ..., z_{li}\}$

We define the dependency of f upon $\{f_i\}$ (and therefore the resulting hierarchy) as a function g:

$$g : Z^l \rightarrow \cup_i (Z^{li}) | F^k \cup_i (F^{ki}) \tag{5}$$

Simply said, a function of any number of *variable* terms from SLA f equals a function of any number of *variable* terms from one or more SLAs f_i, under the circumstances defined by the relevant fact sets. Operating under this highly abstract definition allows us the required flexibility to describe contracts with dependencies of any kind, as long as each of them does eventually evaluate to either *true*, or *false*.

4.3 BDD Mapping

We now have a formal representation of SLAs (Equation 4) and SLA dependencies (Equation 5). The gain in using BDDs lies in *reduction*. Through this process, a BDD becomes a *canonical* representation of the boolean function it describes, as proven in [1]. Therefore, a SLA described as a boolean function in the form of a BDD takes a unique, well-specified and minimal form, eliminating redundancy and allowing to make the mapping which describes SLA dependencies far more efficient than what it would be if we operated on complete graphs. Additionally, the canonical form of the SLAs allows objective evaluation and comparison for maximizing utility.

The exact method to construct a BDD from a SLA depends on the format in which this SLA is originally expressed, and therefore it cannot be algorithmically defined in a universal way. In the case of WS-Agreement we would use the

`Context` and `Service Description Terms` as facts; `Qualifying Conditions` as conditions; `Guarantee Terms` as clauses; and `Term Compositor Terms` could be classified as either conditions or clauses. In fact, WS-Agreement's Term Compositor Terms are essentially boolean operators: `All` (AND), `OneOrMore` (OR), `ExactlyOne` (XOR). Using this pre-defined knowledge for such a specific SLA language, it is straightforward to implement a parser that can read the documents and construct a (Reduced Ordered) BDD on-the-fly as described in [12] with the revised "APPLY" operation.

To illustrate the reduced form of BDDs representing SLAs, we will use the example scenario from Section 4.1. As mentioned, Equations 2 and 3 represent the two example SLAs as boolean functions of the variables from Table 1. Then, assuming an ordering corresponding to the numbering of the variables, the two resulting BDDs would be as in Figure 3.

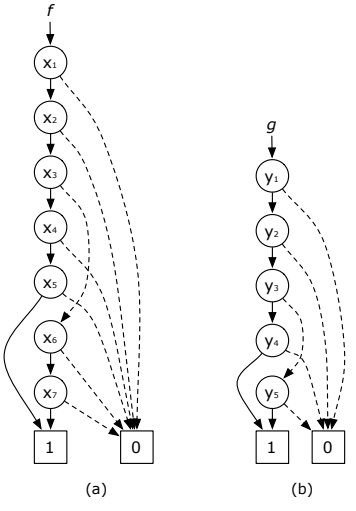

Fig. 3. The BDDs corresponding to functions from Equations 2 and 3

The main deficiency of BDDs is their reliance on the ordering of the variables. The size of a BDD for the same function may vary from linear to exponential, depending on how variables are ordered [1]. Generic algorithms for near-optimal orderings of variables during or after BDD construction have been researched extensively in the past (e.g. [13,14]). Our application to the domain of SLA management and the involvement of *facts* as variables, whose *else* edge always points directly to terminal node **0**, provides already a possibility for optimizing the BDD by pushing all facts to the top of the diagram. Although this kind of ordering does not reduce the total number of nodes, it allows us to ensure that indisputable facts are honored by all parts of the SLA, otherwise it will evaluate to false at runtime (i.e. it is violated). Also, at negotiation time, this ordering

may speed up the negotiation process significantly, since the first thing to be confirmed as acceptable (or not) is the agreement of the involved parties on the essentials of the contract (for instance, monetary unit). It should be underlined, at this point, that facts are propositions which apply to the complete contract, and govern all terms included. Therefore, in certain cases, additional attention is required for choosing what is a fact and what is not. Let us consider, for instance, the case of a two-party contract with two sections describing the obligations of each party, starting each section with an indication as to which party it applies. The statement "section (a) describes the obligations of party (A)" is certainly true for the complete contract. Nevertheless, if reference to the section includes some contract-locality constraint, e.g. "*this* section describes the obligations of party (A)", then this causes ambiguity and cannot apply to the whole contract any more – therefore should be modeled as a *condition*.

Having ordered *facts* at the beginning of the diagram, we assume some BDD method to optimize the ordering of *conditions* and *clauses*. Additionally to generic methods described in relevant literature, a kind of structural optimization that takes advantage of the semantics of SLAs and may be applied here is one that considers what is more crucial to the user. Certain SLA representations contain sections on *Business Values*, that may reference specific terms as regards their importance. Given proper formalization of such sections, a constructor of BDDs from SLAs can take them into account and order clause variables from maximum to minimum importance, thus allowing faster evaluation of business-critical terms.

We can now discuss principles for the SLA application domain, and for outsourcing parts or all of the contract. Starting from the very semantics of SLAs represented as BDDs, we have to distinguish between the meaning of a boolean variable (and the whole diagram) during negotiation time, and during runtime.

4.4 Negotiation Time Operations

During negotiation time, the evaluation of a fact variable to *true* or *false* shows whether the fact is recognized as such from the receiving party. For conditions and clauses, it indicates whether there is any reachable state based on assignments of respective variables, so that the condition / clause under examination can eventually evaluate to *true*. Extending this to the complete diagram, at negotiation time we are interested to see if there exist, in general, truth assignments for the whole set $F^k \cup Z^l$ which satisfy the diagram and lead to **1**. At this point lies an implicit decision. The party that receives the offer needs to have some certainty that it can honor it after signing. It is a policy issue if this certainty needs to be 100%, or near that, or even much lower (perhaps indicating a high-risk strategy). Whatever the policy, the decision will have to be taken based on some objective criteria. A certainty of 100% would mean that paths of the BDD must be checked for *tautology*, that is, any truth assignment for a path will lead to terminal node **1**. If tautology applies for a single path, that should be enough to accept the offer. If not, it is necessary to make an educated guess whether the offer is acceptable, and whether some part needs to be subcontracted.

A simple calculation that can be performed, is the following: Let Γ_f^1 be the set of all paths for f that connect the root to terminal node **1**, and Γ_f^0 the respective set of paths leading to terminal node **0**. We assume that by means of historical monitoring information, forecasting, or simply common sense (e.g. time of day) there is assigned to each node u_i in $h \in \Gamma_f^1$ a probability $P'(u_i)$ to evaluate to a result so that node u_{i+1} is (also) on the same path, and $1 - p$ to evaluate otherwise. If the variables of the nodes in the path are *dependent*, then we need to take this into account and calculate the *conditional probability* of each variable, given the evaluation of all previous variables on this path:

$$P(u_{i+1}) = P'(u_{i+1}|u_1 \cap u_2 \cap ... \cap u_i) \tag{6}$$

In somewhat less formal notation, we have represented the variables (and the events of them taking a value of *true* or *false*) by the names of their nodes. If the variables are *independent*, then $P(u_i) = P'(u_i)$, $\forall i$. The probability p_h that the complete path evaluates to *true*, is

$$p_h = \prod_u P(u)|u \in h \tag{7}$$

Then, the total probability that the SLA can be honored if established, is

$$C = \sum_h (p_h)|h \in \Gamma_f^1 \tag{8}$$

Assuming that the acquisition of this probability per node can be performed in constant time, then the complexity of estimating this probability per path is $O(n)$. The consequent requirement to minimize the total number of paths at construction time or variable ordering time, should also be taken into account.

A negotiating party will want C to exceed some threshold, in order to agree to an offer that was received. If this is not the case, then the party (typically, a service provider) will have to either reject the offer, or try to increase C by subcontracting one or more paths and thus increasing their contribution to the total success probability. Representing SLAs as BDDs is most useful at this point: The canonical and reduced form of a BDD produces a tractable list of options with regard to what we can assign to subcontractors. For items in such a list, due to the specific ordering of variables, we can devise unique and unambiguous signatures. The latter may then be associated to different boolean functions, which represent candidate subcontracts. Domain-specific intelligence can be applied by area experts before operation starts, and define the dependencies of certain variables on others for subcontracts. Then, a system based on these principles can make use of this knowledge, and construct proper offers towards third parties. As long as these offers are accepted, and the respective second-level SLAs are established, it should be the case that the corresponding path has increased certainty to complete successfully as regards honoring the first-level SLA. The negotiating party has a choice, according to policies and strategies, to modify the offer and return it with specific values for the variables of that path (practically

suggesting the SLA equivalent of the path), or to accept the complete SLA as long as the increase in C is sufficient.

Coming back to the example scenario from Section 4.1, we can see two possible ways where this kind of subcontracting is / may be needed. The first, is the explicitly mentioned subcontracting from the SaaS to the IaaS provider. Conceptually, since the SaaS provider has no infrastructure, they cannot offer the service at all unless they subcontract for infrastructure. Terms x_4, x_5, x_6 and x_7 would always evaluate to *false* unless infrastructure resources are available for the software to execute on. As such, the SaaS provider has to go through this translation process in any case, to calculate infrastructure requirements and make a respective offer to the IaaS provider. If an agreement with the IaaS provider already exists, the contracting system in use should find this automatically after the translation occurs, try to reuse it if possible, otherwise resolve to making a new offer. It must be noted here that, since the outsourcing concerns *paths*, the SaaS provider may just as well make two different offers to two different infrastructure providers (one for each of the two paths in Γ_f^1), or can make a single offer to one infrastructure provider for both paths (this is our assumption in the example scenario).

The second case, is if it so happens that the IaaS provider cannot satisfy the incoming offer – for instance, does not have the resources to offer the requested performance during business hours. This means that, according to its estimation, y_4 would evaluate to *false* most of the times, and therefore path $y_1 - y_2 - y_3 - y_4 - \mathbf{1}$ would contribute minimally or not at all to the whole agreement's C-value. In this case, the IaaS provider can reject the offer, or —depending on projected utility— try to outsource this path to another IaaS provider. Further translation of the terms may occur or not in this case, depending on the structural and qualitative agreement properties that are accepted by the second IaaS provider.

It should be mentioned that an offer may be for a single SLA, or for multiple SLAs (typically for different services or groups of services) in the form of a Shared BDD. Our working assumption of an offer for a single SLA does not affect generality.

Another relevant point is that we are referring to SLA terms in a most abstract way, and that is on purpose in order to define a generic model. However, from an implementation point of view, we need to define proper *term signatures* (term templates), and to select "good" values to replace in them. For example, the expression "*completion time* $<$ *5 seconds*" evaluates to true or false and therefore can be modeled as a single boolean variable. Yet, if we assume that the expression "*completion time* $<$ *4 seconds*" is a term with a different signature, then naturally the complexity of mapping between different signatures increases enough to make the problem unfeasible. Therefore, from an implementation point of view, we need a single signature like "*completion time* $<$ *duration*", allowing to set duration to a preferred ("good") value as mentioned before. Here, "good" has to do with the notion that there is some utility coming out of each SLA, and this utility we wish to maximize. Structural optimization of the SLA's BDD supports better decisions from a SLA computability point of view, and possibly reduces time to reach an agreement. However, the utility

itself is domain-specific again, and falls into the same realm with choosing a "SLA probability to succeed" threshold over which an offer is acceptable.

Solutions to the open issues elaborated in the previous paragraph are outside the scope of the work presented in this paper. *Technology mapping* [15,16] is a concept which matches the problem of templating terms and their combinations, and provides a starting point for further research. The topic of selecting values that increase total utility falls under *multi-objective optimization* [17]. As a matter of fact, the optimization logic may affect the negotiation process itself. An entity negotiating over a set of variables may find that small modifications to the negotiating party's requirements may increase significantly the resulting utility. In this case, it may just as well modify the proposed term slightly, and return a counter-offer which does not match the other entity's requirements, but may provide much better results if accepted. Such negotiation-time risk-taking attitudes can be modeled with *game theory* methods [18,19]. Technologies from all three areas will be tested in the future as part of this work and a complete implementation.

4.5 Runtime Operations

For this part of our work, we are assuming a monitoring subsystem that can capture service execution-related events from various sources and detect if some SLA term is being violated. The process actually starts much earlier, during negotiation. At that time already, we must verify that terms of an agreement *can* actually be monitored [20]. Following this verification step, as part of the negotiation process, a SLA may be formally established, perhaps relying on other SLAs for its existence.

While the service is being consumed, incoming events are processed and terms (in the form of boolean variables of the BDD) are examined to see if a violation has occurred. The ordering of the variables allows the linear-time confirmation, starting from the root and traversing the diagram towards terminal nodes. As each variable evaluates to *true* or *false*, the respective child (*then*/*else*) is followed until a terminal node is reached. If that node is **0**, then there exists a violation, and the reason of failing at that specific part of the SLA must be assessed. Depending on whether this failure happened on a path which was outsourced, or not, there may be a re-negotiation initiated, penalties claimed, or simply adjust the method to estimate success probabilities for different paths. Additionally to corrective actions, such an event must be logged to be reused in next negotiation cycles.

The exact methodology to use in order to avoid unnecessary evaluations of the complete diagram, depends on the monitoring system, the way to evaluate each variable, and the acceptable time thresholds for reaction to violations. A complete definition of such methodologies is out of scope for this work.

5 Experimental Verification

As a proof of concept, we built a very simple prototype that accepts a SLA already expressed as a boolean function in *Reverse Polish Notation* (RPN) form,

produces a BDD from it and assigns probabilities to the nodes in a semi-random way. Then, it calculates the paths leading to **1**, their probabilities to be followed and the total probability that the SLA can be honored without any subcontracting. We experimented with a single SLA offer, which was crafted not to contain dependent variables, according to the following description:

> *The SLA concerns service "Service-1" (*fact*). Business hours are set to 09:00-17:00 (*fact*). The whole system must run in isolation from other customers of the service provider (*fact*). If the operation invoked is "Operation-1" (*condition*), and time is within business hours (*condition*), then: completion time should be less than 5 seconds (*clause*); availability should be more than 99% (*clause*); and throughput should be more than 100 operations per minute (*clause*). For times outside business hours: completion time should be less than 10 seconds, availability should be more than 95%, and throughput should be more than 50 operations per second. For operations other than "Operation-1", invocations should be authenticated (*clause*) and availability should be more than 98%.*

With regard to the assignment of probabilities to the nodes and the paths to follow, we assigned a probability equal to 1.0 to facts and to the proposition of authenticated invocations (this being a functional requirement that the provider should be aware of). We then assumed that invocations of "Operation-1" are one out of three, i.e. a probability of roughly 0.33, and the same for the time of day being within business hours – so we imply that invocations of the service are equally distributed throughout the day. Finally, for the propositions of completion time, availability and throughput, we randomly assigned on each node a probability between 0.8 and 1.0 that the provider can satisfy it or will fail (a second random number indicates which of the two applies). In a real scenario, the provider would calculate these probabilities based on monitoring, forecasting or other information. Eventually, we run this simple scenario 10000 times, to see under these semi-random conditions how the SLA success estimations behave. Constructing the BDD for this specific SLA took place in a mere 2.2 seconds. Running the 10000 probability tests took approximately 4 seconds on a 2.4 GHz processor. The diagram contained 16 levels, excluding terminal nodes. Of the 21 paths leading to terminal node **1**, the shortest was 6-nodes long (excluding **1**), and the longest was 13-nodes long. Figure 4 illustrates the overall calculated probability that the SLA will be successful if established.

From this preliminary evaluation, the feasibility and validity of the approach is exhibited for all SLAs that consist of propositions evaluating to *true* or *false*. As long as all invariable statements of a SLA (e.g. references to other SLAs) can be expressed as facts, and all variable statements can be expressed as conditions and clauses, this assumption is valid for any SLA. In this experiment, a simple but not trivial expression was built fairly quickly, producing a vector of 21 paths to evaluate and monitor. Allowing some certainty for individual terms (80%-100% probability of success or failure) results in a clear gap between SLAs projected to fail, and those projected to succeed. This is an indication that, using BDDs

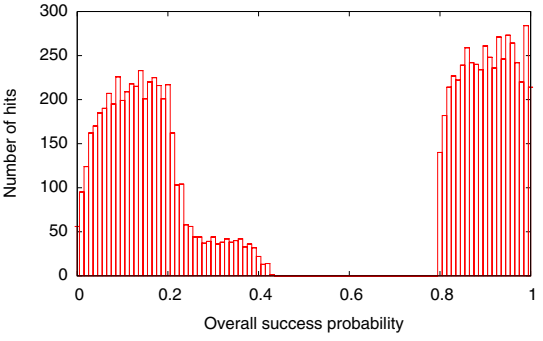

Fig. 4. Experimental result

in this context and under such circumstances, we can calculate in only a few milliseconds and with a reasonable amount of certainty, whether the complete SLA can be satisfied or not. Future application of this methodology on real-world use cases will allow for further evaluation.

6 Conclusions and Future Work

In this paper we presented a novel application of (Shared) Reduced Ordered Binary Decision Diagrams, for representing and managing SLAs, as well as facilitating the construction of SLA hierarchies. BDDs are graph-based structures which have been used for decades in the field of VLSI design and verification, with particular success. They are one of the main tools of the VLSI industry for testing prototypes, and therefore BDDs are a topic under heavy research for decades. The depth and breadth of existing ideas and research can be applied to SLA management for further advancement of this complex service management area. In this particular work we elaborated on the representation through a formal definition of SLAs as boolean functions and from there as BDDs; explained the advantages of this approach; and showed how such kind of use is possible for negotiating SLAs, subcontracting (leading to implicit SLA hierarchies) and detecting SLA violations. Finally, we briefly discussed the encouraging experimental results of applying BDDs to SLA representation.

In the near future we will fully implement these ideas as part of a more general SLA management design. It is our purpose to explore the topic of BDD structural optimization, in addition to that of multi-objective optimization, the latter being necessary for increasing a SLA's utility. Technology mapping appears to fit well the requirement to translate between abstract logic representations, and game theory is suitable for negotiation mechanisms. These technologies will also be evaluated and possibly applied to our implementation.

References

1. Bryant, R.: Graph-Based Algorithms for Boolean Function Manipulation. IEEE Transactions on Computers C-35(8), 677–691 (1986)
2. Lee, C.: Representation of switching circuits by binary decision diagrams. Bell System Technical Journal (38), 985–999 (1959)
3. Akers, S.: Binary Decision Diagrams. IEEE Transactions on Computers C-27(6), 509–516 (1978)
4. Open Grid Forum: Web Services Agreement Specification, WS-Agreement (2007)
5. Keller, A., Ludwig, H.: The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services. Journal of Network and Systems Management 11(1), 57–81 (2003)
6. Binder, W., Constantinescu, I., Faltings, B.: Scalable Automated Service Composition Using a Compact Directory Digest. Database and Expert Systems Applications, 317–326 (2006)
7. Campailla, A., Chaki, S., Clarke, E., Jha, S., Veith, H.: Efficient filtering in publish-subscribe systems using binary decision diagrams. In: ICSE 2001: Proc. 23rd International Conference on Software Engineering, pp. 443–452 (2001)
8. Paschke, A., Bichler, M.: Knowledge representation concepts for automated SLA management. Decision Support Systems 46(1), 187–205 (2008)
9. Ebendt, R., Drechsler, R., Fey, G.: Advanced BDD optimization. Springer, Heidelberg (2005)
10. Shannon, C.E.: A symbolic analysis of relay and switching circuits. AIEE (57), 713–723 (1938)
11. Bhoj, P., Singhal, S., Chutani, S.: SLA management in federated environments. Computer Networks 35(1), 5–24 (2001)
12. Bryant, R.E.: Symbolic Boolean manipulation with ordered binary-decision diagrams. ACM Comput. Surv. 24(3), 293–318 (1992)
13. Friedman, S., Supowit, K.: Finding the optimal variable ordering for binary decision diagrams. IEEE Transactions on Computers 39(5), 710–713 (1990)
14. Rudell, R.: Dynamic variable ordering for ordered binary decision diagrams. In: ICCAD 1993: Proc. 1993 IEEE/ACM international conference on Computer-aided design, pp. 42–47. IEEE Computer Society Press, Los Alamitos (1993)
15. Keutzer, K.: DAGON: Technology Binding and Local Optimization by DAG Matching. In: 24th Conference on Design Automation, June 1987, pp. 341–347 (1987)
16. Detjens, E., Rudell, R., Gannot, G., Wang, A., Sangiovanni-Vincentelli, A.: Technology mapping in MIS. In: Proc. International Conference on Computer Aided Design, November 1987, pp. 116–119 (1987)
17. Ehrgott, M.: Multicriteria Optimization. Springer, Heidelberg (2005)
18. Fatima, S., Wooldridge, M., Jennings, N.: A Comparative Study of Game Theoretic and Evolutionary Models of Bargaining for Software Agents. Artificial Intelligence Review 23(2), 187–205 (2005)
19. Figueroa, C., Figueroa, N., Jofre, A., Sahai, A., Chen, Y., Iyer, S.: A Game Theoretic Framework for SLA Negotiation. Technical report, HP Laboratories (2008)
20. Comuzzi, M., Kotsokalis, C., Spanoudakis, G., Yahyapour, R.: Establishing and Monitoring SLAs in Complex Service Based Systems. In: ICWS 2009: Proceedings of the 2009 IEEE International Conference on Web Services, pp. 783–790 (2009)