2009

Automata-Based Verification of Non-Functional Requirements in Web Service Composition

Hongyu Sun
Iowa State University

Automata-Based Verification of Non-Functional Requirements in Web Service Composition

Hongyu Sun[1] , Samik Basu[1] , Robyn Lutz[1,2] and Vasant Honavar[1]
[1]*Department of Computer Science, Iowa State University*
Ames, IA, 50011-1040, USA
[2]*Jet Propulsion Laboratory/Caltech*
{sun, sbasu, rlutz, honavar}@cs.iastate.edu

Abstract

We address the problem of how to provide guarantees to a user that an automatically generated composition of independently developed web services meets the non-functional requirements (NFR). The user-specified NFR are in the form of hard constraints. We introduce an automata-based model for representing and reasoning about non-functional requirements for verifying the conformance to NFR. The approach described here enables this verification by lifting the NFR analysis from the level of individual services to the level of the search space of candidate compositions obtained from the functional requirements. The proposed approach can accommodate the different subsets of NFR for different components of a composite service. We introduce three different strategies when multiple NFRs exist and analyze their relative advantages and disadvantages under different scenarios. We present results which show that this approach to verifying the NFR can support efficient re-verification of web-service compositions whenever NFR are updated. The approach described here has been applied in service composition based on NFR in an Emergency Management System.

1. Introduction

While a variety of practical mechanisms has been deployed for synthesizing composite services taking into consideration their desired functional requirements [1][2], the handling of non-functional requirements (NFR) is still largely ad-hoc, informal and difficult to verify. Among the difficulties are that NFRs may be ambiguously defined by the user or that there may be inconsistencies among the NFRs specified by the user.

While functional requirements describe how a system should behave during operation, NFRs describe constraints on the quality attributes of the system's operation. NFR can be broadly classified as soft and hard constraints. Hard constraints refer to the set of NFRs that must be satisfied by a composite service, while soft constraints deal with user preferences and trade-offs among NFRs [3].

Automata-based NFR Representation and Analysis. This paper shows how an automatically generated composition of independently developed web service can be shown to meet the NFRs imposed by the user as hard constraints. The approach described here enables this verification by lifting the NFR analysis from the level of individual services to the level of the search space of candidate compositions obtained from the functional requirements.

The verification process introduced in this paper, assumes the existence of a search space containing all the candidate service compositions that satisfy the functional requirements. The functional composition algorithms can be found in [9] and will not be described here.

In our setting, a composition of services is represented as an automaton where states represent the services participating in the composition, inter-state transitions represent composition between the services at the source and destination states, and labels on transitions represent the predicate set of all the non-functional attributes of the services connected via the transition.

Automata are also used to represent NFRs. The predicates in the NFR automata are set true by the participating services during the verification process.

The problem of whether a composition conforms to a desired NFR is addressed by synchronously composing the automaton representing the service composition with the automaton representing the corresponding NFR. The composition is said to satisfy the NFR if and only if the sequence of properties over non-functional attributes as represented by the NFR-automata is also present in the automaton of the composition.

Introducing Scopes in NFR Automata. In many situations, a user imposes a certain NFR not on the entire composition but only on a subset of the services participating in the composition. To accommodate different subsets of NFRs for different components of a composite service we introduce the notion of scoping. To achieve this we incorporate auxiliary scoping information in the property automata as described in Sect. 4.1.

Liveness as Safety NFRs. An automata-based property model can represent both safety and liveness properties [4][5], where a safety property is of the form "a program never enters an undesirable state" and a liveness property is of the form "a program eventually enters a desirable state". To achieve compositional verification of the NFR properties we introduce a way of handling liveness properties by converting them to safety properties via use of an additional *trap* state, described in Section 4.2.

If a state violating a safety property is encountered during composition or if a state satisfying a liveness property is not reached in any branch, we say that the current candidate service composition in the search space violates the NFR. *Verification Strategies.* We also present three different strategies (Sect. 5) for verifying the conformance when multiple NFRs exist and analyzing their relative advantages and disadvantages. Specifically, the three strategies are: *independent composition (INC)*, where the product of each NFR-automaton and the composition automaton is performed separately; *big-bang composition (BBC)*, where the product of the NFR-automata and the composition automaton is computed in one shot; and *two-stage composition (TSC)*, where the product of all the NFR-automata is first computed and stored, and only then is the product of the combined properties and the composition automaton computed.

We show that different strategies are helpful in different composition scenarios, e.g. to maximize reuse or to quickly identify inconsistencies in the NFRs. We show how to analyze the trade-offs among the three strategies before verification by pre-computing which strategy will be most efficient for specific sets of properties. For example, we show that INC and BBC are more efficient when verifying large and complex properties for few candidate service compositions in the search space. TSC is more efficient for handling small, simple properties with a large number of candidate service compositions since it maximizes reuse across the verification process.

Case Study. We apply the approach described here to the NFRs of an Emergency Management System (EMS) [6] example to illustrate the technique and the different verification strategies. Results presented here show that our approach to verifying the NFRs can reuse intermediate results to re-verify web-service compositions as the NFRs are updated. We lift the NFR analysis from the level of individual services during composition time to the level of the search space of candidate compositions obtained from the functional requirements. With this separation of concerns, we are able to separate the NFR analysis from the initial functional web service composition. This separation both simplifies the analysis of the EMS and allows reusability of intermediate results during composition since the functional requirements and NFRs can evolve independently.

Such separation of functional and nonfunctional requirements is also useful because for most commercial web services the functional requirements remain fairly constant while the non-functional requirements may be different for each individual customer. In this case, all candidate compositions matching the functional requirements can be pre-computed and stored in a search space. When a new NFR(s) is provided by a new customer, or updated by an existing customer, the service provider is able to identify which of the candidate compositions already known to satisfy the functional requirements also satisfy the customer's new NFRs.

The **contributions** of this paper are:

1. The introduction of an automata-based non-functional requirements model for verifying web service compositions.
2. The integration of scoping information into the automata model to handle different subsets of NFRs and the introduction of the trap state to verify liveness properties.
3. The presentation and tradeoff analysis of three alternative strategies for verifying multiple properties.

The rest of the paper is organized as following. Section 2 describes related work on web service composition and nonfunctional requirements. Section 3 gives an overview of our approach and introduces the EMS illustrative example used in the rest of the paper. Section 4 and Section 5 show how to model and verify the NFR properties, respectively. Section 6 discusses the verification of multiple properties by means of the three alternative strategies. Section 7 applies our approach to an excerpt from the EMS. Section 8 provides a brief conclusion.

2. Related Work

Most web service composition approaches focus on satisfying the user's functional requirements (see, e.g., [1][2]). The work reported here builds on model-based verification to also consider NFRs. Foster, Uchitel, Magee and Kramer describe a model-based approach

(LTSA-WS) to verify composition implementations [7]. The verification mechanism is a trace equivalence check, which composes labeled transition systems (LTS) based on pre-constructed finite state process models. Rao, Kungas and Matskin introduce a method for semantic web service composition based on Linear Logic theorem proving [8]. All functional and non-functional requirements are translated into propositions in the logical axioms and then processed by theorem prover. However, the Linear Logic prover cannot guarantee acceptable computation efficiency when there are a large number of functional/non-functional constraints on the required system.

We build on previous work by Pathak, Basu and Honavar, who introduced a tool-supported approach (MoSCoE) for composing new services with existing ones according to the user requirements [9]. The goal of the composition and the component services are all represented as LTS automata. A forward-backward web service composition algorithm constructs a service composition that satisfies the goal automata by selecting appropriate component services in the repository. It considers some non-functional constraints but makes several simplifying assumptions. First, all non-functional constraints must be represented by a single threshold. Second, only non-functional constraints that can be calculated by arithmetic computations are handled (e.g., overall cost of the services or total service delay). These assumptions limit the power of the algorithm to handle NFRs. In this paper we model the NFRs as automata and verify both non-functional properties and functional behavior through automata composition.

Several researchers have discussed NFRs in the context of Quality of Service (QoS). Jaeger, Rojec-Goldmann and Muhl describe QoS aggregation for web service composition using workflow patterns [10]. Zeng et al. use a QoS model with linear programming techniques to achieve the optimal dynamic service composition against the QoS requirements [11]. However, QoS approaches cannot handle broader NFRs. Another approach to composite web service verification is context-based matching, including syntactic matching and semantic matching, in order to ensure compliance of the composition to the functional and non-functional requirements [12][13].

Existing approaches to NFR in web service composition are limited in their representational power and the efficiency of the verification. Our technique tries to improve the representational power by modeling NFRs in automata and tries to improve the efficiency of verifying multiple properties by pre-selecting the optimal composition strategy.

3. Overview

Figure 1 provides an overview of our NFR verification process.

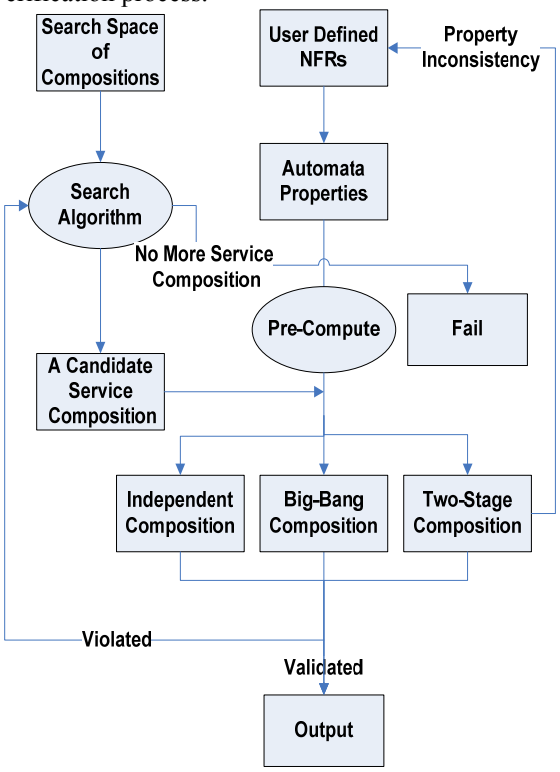

Figure 1. Process diagram of NFR verification

We first translate the user-defined NFRs into formal automata properties and incorporate the scoping information (described in 4.1). We then convert liveness properties into safety properties to allow a uniform verification process. After that, we pre-compute which of several alternate verification strategies will fit the properties better. With the chosen verification algorithm, we then verify a candidate service composition from the pre-constructed search space of compositions meeting the functional requirements against the NFRs. If the verification fails, we find the next candidate service composition and iterate. If we run out of all service compositions in the search space, we report to the user that the verification failed, i.e., that the NFRs are not satisfied by any composition in the search space. If the candidate service composition is successfully verified by the algorithm, we output this candidate as the final result to the user.

We constructed a small system, the Emergency Management System (EMS), based on [6], to illustrate the basic concepts of our approach. EMS consists of several different units: the Field Officer Service, the Request Dispatch Service (Dispatcher for short) and

services for emergency handling, including Ambulance Dispatch Service, Fire Truck Dispatch Service and Police Dispatch Service. The functional requirements are to dispatch ambulance(s), fire truck(s) and police to a location upon request. These requests are specified and sent by the Field Officer through a service in a mobile terminal or a PDA. The Dispatcher then assigns the request an electronic certificate. With this certificate, the requested ambulances and police are authorized for dispatch upon availability by the Field Officer's communicating with the services of the corresponding departments. Two NFRs used in this example are:

1. Every selected Ambulance Dispatch Service should have an average response time of less than ten minutes (Figure 2 shows this NFR).
2. When a Field Officer requests an ambulance, eventually an ambulance from the same jurisdiction (city, county) as the Field Officer shall be summoned.

4. Property Modeling

4.1 Automaton Extension for Property Modeling

Definition 1: A finite state automata is a tuple FSA = (S, s_0, Δ, P, F) where S is the finite set of states, $s_0 \in S$ is the start state, and $\Delta \subseteq S \times 2^P \times S$ is the transition relation of the form $s - \phi -> s'$ such that $s, s' \in S$, and $\phi \in 2^P$ is a subset of propositions P. Finally, $F \subseteq S$ is the set of final states. A finite sequence is said to be accepted by the FSA if and only if the sequence starts from s_0 and terminates in any of the final states in F. NFR and compositions are described using FSA.

A property specification pattern is a generalized description of a commonly occurring requirement on the sequences of permissible propositions on the transitions of a finite–state model [14]. However, in the web service composition, a NFR may refer only to the properties or behaviors of some services. For example, in the EMS, the Field Officer may only impose a response time requirement on the ambulance but not on the police.

We incorporate scoping information into the automata properties to handle such situations. A *scope* of a property is a user-defined subset of the services and their interactions. The extension introduces a special guard over source states or target states using the reference keyword "Source" or "Target". The transitions with such guards are only enabled when the source (respectively, target) state satisfies the guard.

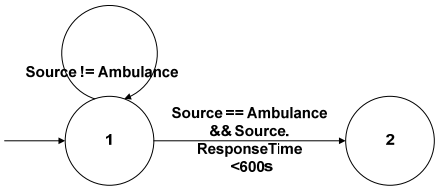

Figure 2. Transition guard with constraint on the service attributes

4.2 Efficient Analysis of Liveness as Safety Properties

We are concerned with the representation of both safety properties and liveness properties. A safety property is of the form "a program never enters an undesirable state" and a liveness property of the form "a program eventually enters a desirable state" [4][5]. The undesirable state in the safety property is labeled as a *trap state*, π [4]. In Section 3, the first NFR is a safety property and the second NFR is a liveness property.

Conversion of liveness properties to safety properties via a trap state. We introduce a new way to uniformly handle properties by converting liveness properties to safety properties via use of an additional trap state. We introduce a termination message "#T" which indicates the end of the service composition in the search space. A liveness property automaton can be translated into an equivalent safety property automaton, given the termination message "#T". For example, for the liveness property in the second NFR in Sect. 3, we translate it into a safety property (Figure 3) by creating a trap state that captures all the executions that have not reached the termination state when the service ends.

The transitions in the property automaton shown in Figure 3 are triggered by the true predicates that appear in the service composition. The property automaton can only be initiated by the Field Officer Service in State 1. When it detects this service, it records the location of the "Officer" and goes to State 2. State 2 can be triggered by the service Ambulance Dispatch Service and requires that its location be the same (i.e., the same jurisdiction) as the recorded location for the Field Officer. At this point there are two cases. If the automaton reaches the end State 3, the property is successfully verified for the target service composition. If the service composition detects #T and the automaton has reached the trap state, the verification has failed.

To summarize, the verification algorithm consists of the following stages. Given a candidate service composition and a property automaton:

1. Convert every liveness property into a safety property.
2. Calculate the product of the service composition automaton and the property automaton.
3. If a trap state in a safety property is reached during composition, return failure. Otherwise return success.

We next describe the composition algorithms.

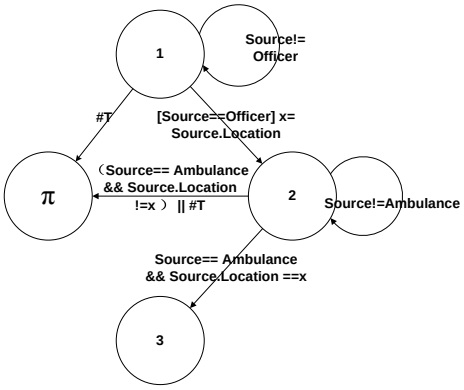

Figure 3. Sample safety property for EMS

5. Automata Composition

In our approach all automata compositions are synchronous, i.e., multiple automata can make progress in parallel for each step [15]. The problem of whether a composition conforms to a desired NFR is addressed by calculating the synchronous product of automata representing the composition with those representing the corresponding NFR. The composition is said to satisfy the NFR if and only if the sequence of properties over non-functional attributes as represented by the NFR-automata is also present in the automaton of the composition. The verification process can be viewed as an equivalence check.

The left side of Figure 4 shows the search space of a service composition. Each branch represents a different candidate service composition under the functional requirement constraints, with each state being a component web service. The right side of Figure 4 is an NFR property automaton. The guards on the transitions in both sides represent subsets of the propositions. The difference is that a proposition subset on a transition in the search space automaton on the left represents all true predicates related to the source service and target service of the transition. On the other hand, the proposition subset on a transition in the NFR property on the right represents all the user-specified non-functional constraints which are pending to verify.

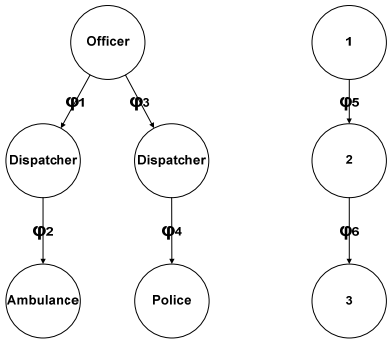

Figure 4. The search space and the NFR property

Definition 2: Given two automata, $FSA_i = (S_i, s_{0i}, \Delta_i, P_i, F_i)$, for $i \in \{1, 2\}$, their product is another FSA denoted by $FSA_1 \times FSA_2 = (S_{12}, s_{012}, \Delta_{12}, P_{12}, F_{12})$, where $S_{12} \subseteq S_1 \times S_2$, $s_{012} = (s_{01}, s_{02})$, $P_{12} = P_1 \cup P_2$, $F_{12} = \{(s_1, s_2) \mid s_1 \in F_1, s_2 \in F_2\}$. Finally, $s_1 - \phi_1 -> s'_1 \in \Delta_1$ and $s_2 - \phi_2 -> s'_2 \in \Delta_2$ and $(s_1, s_2) - \phi_1 \wedge \phi_2 -> (s'_1, s'_2) \in \Delta_{12}$. Taking the left branch of the left side of Figure 4 as the candidate composition, the product of this candidate composition and the NFR property is shown in Figure 5.

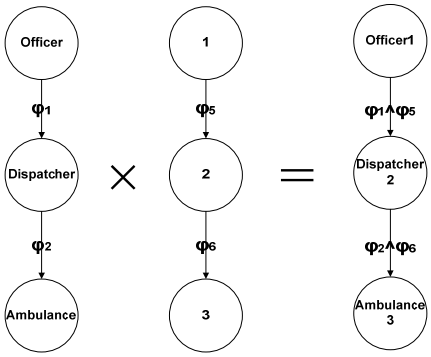

Figure 5. Product of two automata

6. Handling Multiple Properties

6.1 Verifying Multiple Properties

In real-world web service composition, there often exists more than one user specified non-functional requirements. An interesting issue is how to handle the multiple property automata efficiently. Given a candidate composition from the search space, the most straight-forward way to verify each property automata is to verify the properties independently one by one. This strategy, Independent Composition (INC), is shown in the first column in Table 1. A disadvantage of INC is that even if the properties are inconsistent, such that no composition can satisfy them all, the algorithm has to explore the entire search space before telling the user that no composition satisfies the NFR.

Table 1: Three algorithms of verification on multiple properties

Algorithm	Independent Composition	Two-Stage Composition	Big-Bang Composition
Pseudo-Code	1. For each candidate composition s_i in S 1) For each property p_j in P 2) Verify this property by calculating $s_i \times p_j$ 3) If the verification failed, jump to the next s_i 4) End For Each 2. Output s_i and terminate 3. End For Each 4. Output "No composition satisfies the NFR"	1. Take automata $q = p_0$ 2. For j = 1 to k 3. Calculate $q \times p_j$ and Save the result as q 4. If there exists only one termination state in q and it's a trap state in the safety property, Output "Property p_j is Inconsistent with the other properties" and Terminate 5. End For 6. For each candidate composition s_i in S 7. Verify the property by calculating $s_i \times q$ 8. If the verification failed, jump to the next s_i 9. Else Output s_i and terminate 10. End For Each 11. Output "No composition satisfies the NFR	1. For each candidate composition s_i in S 2. Verify the property by calculating $s_i \times p_1 \times p_2 \times \ldots \times p_n$. All automata are composed synchronously rather than pair-wisely. When a trap state is reached, the composition terminates and returns failure. 3. If the verification failed, try the next s_i 4. Else Output s_i and terminate 5. End For Each 6. Output "No composition satisfies the NFR"
Worst Complexity	$O(\sum_i \sum_j (\lvert s_i \rvert \times \lvert p_j \rvert))$	$O(\sum_1^{i=n}(\lvert s_i \rvert \times (\prod_1^{j=k} \lvert p_j \rvert)))$	$O(\sum_1^{i=n}(\lvert s_i \rvert \times \prod_1^{j=k} \lvert p_j \rvert))$
S is the set of all compositions. $S = \{s_1, s_2, s_3, \ldots..s_n\}$ n is the number of compositions in the search space P is the set of all NFRs. $P = \{p_1, p_2, p_3, \ldots..p_n\}$ k is the number of NFRs			

To overcome this weakness of the INC algorithm, we introduce the Two-Stage Composition (TSC) algorithm (the second column of Table 1) which detects property inconsistency before performing the verification. TSC composes all property automata into one combined property automaton prior to verification. In this way, the first inconsistency found during property composition will terminate the verification process and the property automata causing this failure will be captured and returned to the user for possible modification of the NFR.

INC calculates each property independently with the candidate composition and discards the intermediate calculation results after each inner loop. TSC is instead designed to reuse the combined and consistent properties across candidate compositions during verification.

A third strategy, the Big-Bang Composition (BBC) algorithm, replaces the sequential verification of the properties in INC with parallel verification (the third column in Table 1). BBC composes all the automata including the candidate service composition and the property automata synchronously so that the earliest reachable trap state in any of the properties can terminate the verification by returning a failure.

6.2 Comparative Evaluation

For one candidate service composition s in the search space S, the complexity of the three composition algorithms in terms of the number of states visited in the property automaton p_j is:

Independent Composition (INC): $O(\sum_j(|s| \times |p_j|))$

Two-Stage Composition (TSC): $O(|s| \times \prod_1^{j=k} |p_j|)$

Big-Bang Composition (BBC): $O(|s| \times \prod_1^{j=k} |p_j|)$

INC. To pre-determine which strategy is more efficient to apply, we calculate $\sum_1^{j=k} |p_j|$ and $\prod_1^{j=k} |p_j|$. If the *sum* of the number of states in the property automata is less than the *product* of the number of states in the property automata, we select INC to verify the properties. Otherwise, we select either BBC or TSC. For example, if there are two properties with two and three states, respectively, by calculating $\sum_1^{j=k} |p_j|$ we know the sum of the states is 2+3=5. However, $\prod_1^{j=k} |p_j| = 2 \times 3$ will only give an upper bound. Because the automata composition process prunes unreachable

states and merges the same predicates, the composite automata will always have 6 or fewer states.

TSC. Composing all properties first in TSC has merits in verification. It can identify inconsistent NFRs before verification and quickly locate which property automaton caused the inconsistency. Thus, the user can get quick feedback to help in refining a failed NFR. In addition, TSC can assist with efficient verification. Since property automata may share the same predicates on the transitions, the more predicates shared by the property automata, the fewer states the combined automaton will have. Another advantage of using TSC is that the combined property automata can be reused to verify multiple candidate service compositions. Since the other two algorithms don't reuse the computation results at all, it will be more efficient than the other two algorithms if the number of candidate service compositions is large enough. However, we cannot precisely predict this threshold beforehand.

BBC. For a single candidate service composition automaton, BBC is more efficient than TSC. Composing all the properties ahead of verification as in TSC can result in state explosion in the combined property automaton when properties are not related. Composing the candidate service composition automaton with the property automata, on the other hand, will reduce the number of states by pruning the violated branches during verification and can terminate the verification sooner by reaching the nearest trap state in all properties. BBC is thus appropriate for handling cases where there are only a few candidate compositions with more complex properties.

A combined strategy is for the user to pre-set a constant number *k* of candidate compositions as a switch point for the algorithms. If the verification starts with INC or BBC and fails on candidate compositions *k* times, the verification process switches to use TSC on the rest of the candidate compositions. This constant *k* can be determined by multiple simulations or by historical data on the server of the web service provider.

Generally speaking, INC and BBC focus more on the refinement of the design, which regulates the candidate composition, because they can detect the insufficiencies of the candidate compositions quickly. TSC focuses more on the refinement of the user non-functional requirement, because it can more quickly locate any inconsistencies in NFRs.

7. Application

We applied the approach described here to the EMS example to evaluate which algorithm is best for this application. Figure 6 shows the search space; our goal is to verify a web service composition in it against the two NFRs described in Section 3 (the Time Constraint with two states and the Location Constraint with four states) simultaneously. (The two NFR automata are not shown here due to space limitations.) We then translate the liveness property of the Time Constraint into a safety property automaton with three states by following the steps in section 4.2.

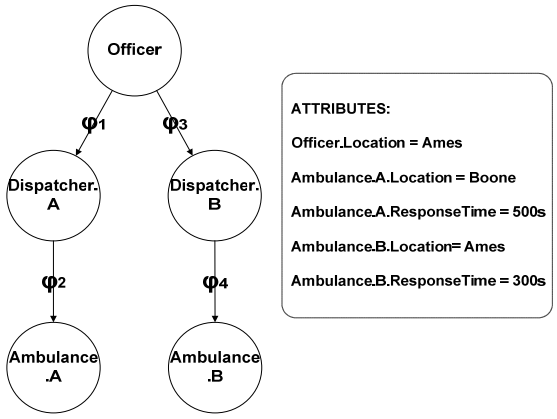

Figure 6. Example search space for EMS application

The two property automata now have three states and four states, respectively, summing to seven states. Figure 7 shows the combined properties (the Time Constraint and the Location Constraint). As we can see, it contains 6 states. Following the process described in 6.1, we pre-compute $\sum_1^{j=k} |p_j| = 7$ and $\prod_1^{j=k} |p_j| = 6$. Since the number of compositions n=2 in this example, the INC algorithm uses 2*3* (3+4) = 42 calculations to verify the two properties whereas TSC uses 2*3*6+3*4=48 calculations (3*4 is the cost of combining the two properties). BBC, on the other hand, requires 2*3*3*4= 72 calculations. With two candidate compositions, applying INC saves 12.5% of the total verification time compared to TSC and 41.6% compared to BBC.

However, if there were more candidates in the search space, the decision is different. For n candidates, applying the INC verification strategy to each candidate composition requires n*3*7 calculation steps while TSC uses n*3*6+3*4 calculations. BBC, on the other hand, requires n*3*4 calculations in the worst case. We can see that, because of the reuse of the combined property automaton in the TSC algorithm, when there are more than four candidates in the search space, TSC will have better efficiency than INC and BBC for this example.

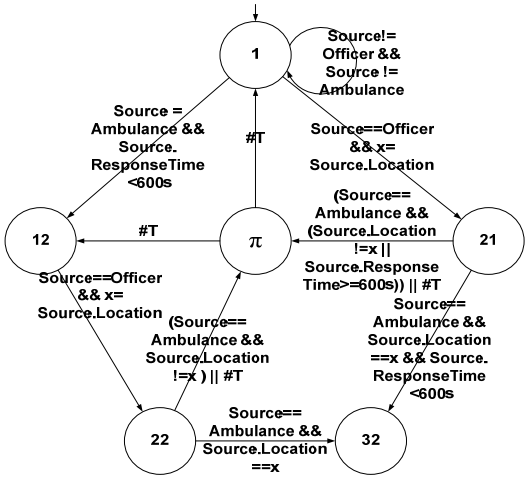

Figure 7. Combined property automaton

8. Conclusion

To enhance verification of non-functional requirements in web service compositions, we have introduced an innovative approach that lifts the NFR analysis from the level of individual services during composition time to the level of the search space of candidate compositions obtained from the functional requirements. By lifting the level of the concern, we separated NFR analysis from functional web service composition. Thus, the specification of NFR for the web service composition is not entangled with the specification of functional requirements. This separation supports a divide-and-conquer strategy in which each stage can be studied separately and then be merged for improved efficiency. Separation also enhances the reusability of intermediate verification results during composition, since the functionalities designed for commercialized web services are much more stable than the users' NFRs.

Our approach to verifying the NFRs can make existing functional web service compositions easier to reuse and re-verify as NFRs are updated. The contributions of this paper include introduction of an automata-based non-functional requirements model for verifying web service compositions, integration of scoping information into the automata model, three different strategies to handle multiple NFRs and a tradeoff analysis of the three strategies for better performance.

9. Acknowledgement

This research is supported by National Science Foundation (NSF) under grant 0541163 and 0702758.

10. References

[1] S. Dustdar and W. Schreiner, "A survey on web services composition", *Int'l Journal of Web and Grid Services*, Vol. 1, No.1, 2005, pp. 1–30

[2] N. Milanovic and M. Malek, "Current Solutions for Web Service Composition", *Internet Computing*, Nov/Dec 2004, Vol. 8, No. 6, pp. 51-59.

[3] L. Chung. B. A. Nixon, E. Yu and J. Mylopoulos, *Non-Functional Requirements in Software Engineering*, Springer, 1999.

[4] S. Cheung and J. Kramer, "Checking Safety Properties Using Compositional Reachability Analysis", *ACM TOSEM*, Vol. 8, No. 1, Jan 1999, pp. 49-78.

[5] H. Guo, Y. Shin and W. Lee, "Enhanced Compositional Safety Analysis for Distributed Embedded Systems using LTS Equivalence", *Proc. 6th ICACS*, Vol. 6, 2007, pp. 115-120.

[6] B. Bruegge and A.H. Dutoit, *Object-oriented Software Engineering: Using UML, Patterns and Java*, Prentice Hall, 2003, pp. 181-196.

[7] H. Foster, S. Uchitel, J. Magee and J.Kramer, "Model-based Verification of Web Service Compositions", *ASE*, 2003, pp. 152-163.

[8] J. Rao, P. Kungas, and M. Matskin, "Logic-based Web services composition: from service description to process model", *ICWS*, 2004, pp. 446-453.

[9] J. Pathak, S. Basu and V. Honavar, "Modeling Web Services by Iterative Reformulation of Functional and Non-Functional Requirements", *4th ICSOC,* 2006, pp. 314-326.

[10] M. C. Jaeger, G. Rojec-Goldmann and G. Muhl, "QoS aggregation for Web service composition using workflow patterns", *Proceedings of the Enterprise Distributed Object Computing Conference*, 2004, pp. 149–159.

[11] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam and Q.Z. Sheng, "Quality driven web services composition", *Proc. 12th Int'l Conference on World Wide Web*, , 2003, pp. 411–421.

[12] K. Zachos and N. Maiden, "Inventing Requirements from Software: An Empirical Investigation with Web Services", *RE '08*. 2008, pp. 145-154

[13] B. Medjahed and Y. Atif, "Context-based matching for Web service composition", *Distributed and Parallel Databases* Vol. 21, No. 1, 2007, pp. 5-37

[14] M. Dwyer, G. Avrunin and J. Corbett, "Patterns in Property Specifications for Finite State Verification", *Proc. of the 21st ICSE,* 1999, pp. 411–420.

[15] M. Huth and M. Ryan, *Logic in Computer Science*, Cambridge University Press, 2004.