A Service Composition Framework Based on Goal-Oriented Requirements Engineering, Model Checking, and Qualitative Preference Analysis*

Zachary J. Oster[1], Syed Adeel Ali[2], Ganesh Ram Santhanam[1], Samik Basu[1], and Partha S. Roop[2]

[1] Department of Computer Science, Iowa State University, Ames, Iowa 50011, USA
{zjoster,gsanthan,sbasu}@iastate.edu
[2] Department of Electrical and Computer Engineering, The University of Auckland, New Zealand
{sali080,p.roop}@auckland.ac.nz

Abstract. To provide an effective service-oriented solution for a business problem by composing existing services, it is necessary to explore all available options for providing the required functionality while considering both the users' preferences between various non-functional properties (NFPs) and any low-level constraints. Existing service composition frameworks often fall short of this ideal, as functional requirements, low-level behavioral constraints, and preferences between non-functional properties are often not considered in one unified framework. We propose a new service composition framework that addresses all three of these aspects by integrating existing techniques in requirements engineering, preference reasoning, and model checking. We prove that any composition produced by our framework provides the required high-level functionality, satisfies all low-level constraints, and is at least as preferred (w.r.t. NFPs) as any other possible composition that fulfills the same requirements. We also apply our framework to examples adapted from the existing service composition literature.

1 Introduction

Service-oriented architectures [8] have become increasingly popular as a way to support rapid development of new applications. These applications may be implemented as *composite services* (also known as *compositions*) that are formed from existing services. The process of developing a composite service that satisfies a given set of user requirements is called *service composition* [16].

Requirements for a service composition may be divided into three main types: functional requirements, behavioral constraints, and non-functional properties. *Functional requirements* describe *what* actions or capabilities are to be provided; for instance, an e-commerce composite service must have a component that handles online payment options. These include both high-level requirements (e.g.,

* This work is supported in part by U.S. National Science Foundation grants CCF0702758 and CCF1143734.

C. Liu et al. (Eds.): ICSOC 2012, LNCS 7636, pp. 283–297, 2012.

the composition must process online payments) and more detailed low-level requirements (e.g., the composition shall verify the identity of credit-card users). *Behavioral constraints* describe *how* the functionality must be provided by specifying required interactions and/or ordering of the component services. For example, the e-commerce composite service must be composed so that the shipping service is not invoked before the payment is confirmed and an address is verified.

Non-functional properties (NFPs) may include quality of service (QoS), cost, scalability, or other desirable properties that are not necessary for the composition to perform the required tasks. Ideally, a composition would satisfy the entire set of NFPs, but in practice trade-offs between NFPs must often be considered. For example, some users might prefer the increased security of locally-hosted services over the greater scalability of cloud-based services. A service composition framework must consider preferences and trade-offs between NFPs in order to identify a composition that satisfies all functional requirements and behavioral constraints while fulfilling an *optimal* set of NFPs.

The Driving Problem. While there exist a number of service composition frameworks and algorithms associated with them (many of which are surveyed in [16]), very few of them consider all of these aspects in a single framework. These existing methods often have one or more other important drawbacks:

- They frequently treat all functional requirements as mandatory, choosing between several versions of the same low-level functionality instead of considering diverse low-level implementations of the same high-level functionality.
- They typically do not focus on verifying low-level behavioral constraints.
- They often consider only NFPs that affect the QoS of the composition but ignore other important NFPs, especially those that are not easily quantified.
- They typically require that the names and/or structures of a service's accepted inputs and available outputs exactly match those of other services.

Our Solution. The *contribution* of our work is a service composition framework that addresses the above shortcomings in the following fashion:

- A **Goal Model** (as used in the Goal-Oriented Requirements Engineering (GORE) [5] methodology) is used to describe the functional requirements for a composite service. Non-functional properties are associated with the nodes of the goal model to indicate how satisfaction of each requirement by an existing service contributes to the satisfaction of the NFPs.
- A **Conditional Importance Preference Network** (CI-net) [3] is used to formally describe qualitative preferences and trade-offs between non-functional properties. We claim that it is more intuitive to express preferences over NFPs in qualitative terms because not all trade-offs may be naturally quantifiable (e.g., it may be difficult or even impossible to describe to what extent locally hosted services are preferred to cloud-based services).
- **Model Checking** is used to automatically construct a composition that satisfies behavioral constraints specified in Computation Tree Logic (CTL) [7]. The final composite service is chosen from a set of preferred candidate compositions that satisfy the overall functional requirements. Structural mismatches between input/output data types are also resolved in this step.

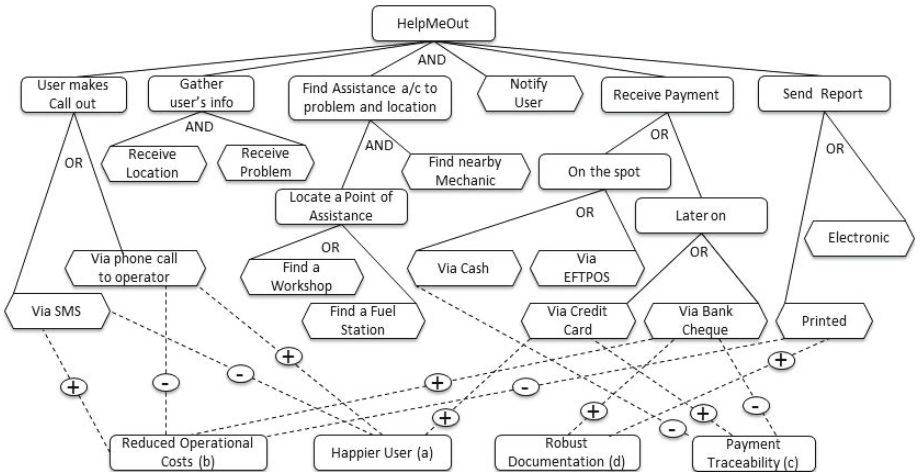

Fig. 1. *HelpMeOut* goal model

We formally *prove the correctness* of the results computed by our framework. We also show our framework's *practical feasibility* by applying it to produce a correct composite service that solves a non-trivial service composition problem.

This work advances the state of the art in service composition by considering high-level functional requirements, low-level behavioral specifications [1], and NFPs [14] all at once, which allows the search space for candidate compositions to be effectively reduced (relative to considering these aspects separately). Although a few existing techniques provide such integrated solutions, our framework handles a wider range of problems than those techniques.

Organization. Section 2 introduces the example used to demonstrate our approach. Section 3 describes the existing concepts that form the basis of our composition framework. Section 4 presents our framework in detail and proves its correctness. Section 5 describes our implementation and results from three case studies. Section 6 discusses related work on similar problems in service composition. Section 7 concludes the paper and discusses future avenues for research.

2 Illustrative Example

We motivate our composition approach using the example of *HelpMeOut*, a proposed service composition (taken and modified from [1]) that makes it easier for a vehicle's driver to call for assistance in case of an emergency.

Functional Requirements. Figure 1 presents the functional requirements for the *HelpMeOut* system using a goal model (AND-OR graph). They include collecting the vehicle's location and problem, searching for a nearby point of assistance, locating a mechanic that can visit the user, receiving payment, and reporting the event. Intermediate requirements appear in round-edged boxes, while basic requirements that may be realized from available services are shown in

hexagons. The graph illustrates dependencies between the requirements. For instance, the root (level 0) describes the overall functional requirement, which is realizable if all of the requirements at level 1 are satisfied (**AND-decomposition**). In contrast, the requirement *Receive Payment* is satisfied if either of the requirements *On the Spot* or *Later On* is satisfied (**OR-decomposition**).

Preferences and Trade-offs. Along with functional requirements, Figure 1 captures dependencies between NFPs and services. These are represented in boxes which are connected to functional requirements via edges annotated with "+" or "-". The "+" annotation represents the satisfaction of the functional requirement having a positive impact on the non-functional property, while the "-" annotation represents a negative impact. For instance, satisfying the requirement that the users can contact an operator via a phone call may result in a happier user (positive impact) but will have a negative impact on reducing operational cost.

It may not be possible to consider a set of basic requirements such that (a) they have only positive impacts on the NFPs, (b) all NFPs are considered, and (c) the root-level requirement is satisfied following the traditional semantics of "and" and "or". Therefore, preferences and trade-offs over NFPs are important for identifying a preferred set of basic requirements that result in satisfying the root-level requirement. Consider the following preference statements:

1. If robust documentation is used, payment traceability is more important than reducing operational costs.
2. If costs are reduced at the expense of customer satisfaction, then using robust documentation takes precedence over ensuring payment traceability.

Behavioral Constraints. While functional requirements describe the necessary functionalities, behavioral constraints ensure correct low-level interaction or ordering of the services participating in the composition. For example, *HelpMeOut* requires that if the *EFTPOS* or the *Cash* service is used for payment on the spot, then a printed report should be sent instead of an electronic report.

Annotated Service Repository. Suppose there is a repository of services that are available for use in the composition. Each service is specified using a standard service specification language such as WSDL [4], which describes the service's high-level functionality (semantics) as well as its inputs, outputs, and low-level behavior. From this complete specification, a *labeled transition system* (LTS), which captures the dynamics of the service, is extracted manually.

LTSs for the services in our repository are depicted in Figure 3. Because the *PhoneCall* and *SMS* services serve only as interfaces between a user and the system, their LTSs are not shown to avoid complexity.

3 Preliminaries

3.1 Goal Model: Decomposition of Functional Requirements

The overall functionality of the composition Θ can be decomposed into a Boolean combination of individual functionalities θ [14]. The relationships between θs in

this Boolean combination can be represented graphically as an *AND-OR* graph $\mathcal{G}^{\Theta}$ such as the one for the *HelpMeOut* service in Figure 1. The basic functional requirements that can be realized from available services are optionally associated with NFPs to denote their positive or negative impact. This combination of decomposition of functional requirements and associations to NFPs is known as a *goal model*. Goal models are key to the Goal-Oriented Requirements Engineering [5] methodology, where they are used for the same purposes.

3.2 CI-Nets: Expressing NFP Preferences

In our framework, the user specifies preferences between different sets of NFPs in a qualitative preference language called *Conditional Importance Networks* (CI-nets) [3]. A CI-net P is a collection of statements of the form $S^+, S^- : S_1 > S_2$, where S^+, S^-, S_1, and S_2 are pairwise disjoint subsets of NFPs. Each statement specifies that if there are two outcomes (two candidate services satisfying all functional requirements) where both satisfy S^+ and none satisfy S^-, then the outcome that satisfies S_1 is preferred to the one that satisfies S_2. The preference order induced by the CI-net follows the semantics of these statements as well as a monotonicity rule, which ensures that an outcome satisfying a set Γ of NFPs is preferred to outcomes that satisfy the set $\Gamma' \subset \Gamma$ (all else being equal).

The semantics of CI-nets is given formally in terms of a *flipping sequence* [3]. Given two outcomes γ and γ', γ' is preferred to γ (denoted by $\gamma' \succ \gamma$) if and only if there exists a sequence of outcomes $\gamma = \gamma_1, \gamma_2, \ldots, \gamma_n = \gamma'$ such that for each $i \in [1, n-1]$, one of the following is true:

- γ_{i+1} satisfies one more NFP than γ_i.
- γ_{i+1} satisfies $S^+ \cup S_1$ and does not satisfy S^-; γ_i satisfies $S^+ \cup S_2$ and does not satisfy S^-; and there exists a CI-net statement $S^+, S^-; S_1 > S_2$.

Deciding whether one outcome is preferred to another (with respect to the CI-net semantics) is referred to as *dominance testing*. It relies on generating an *induced preference graph* (IPG), which represents the partial order between outcomes based on the preference semantics, and then verifying reachability of one outcome from another: γ' is reachable from γ if and only if an improving flipping sequence exists from γ to γ'. In [13], we have presented a model checking-based approach for dominance testing based on preferences expressed in CI-nets.

Example 1. Consider the preferences given in Section 2. The first preference can be expressed as CI-net statement

$$\begin{array}{l}\{\textit{Robust Documentation}\}; \{\} : \\ \{\textit{Payment Traceability}\} > \{\textit{Reduced Operational Costs}\}\end{array} \quad (1)$$

The second preference can be expressed as CI-net statement

$$\begin{array}{l}\{\textit{Reduced Operational Costs}\}; \{\textit{Happier User}\} : \\ \{\textit{Robust Documentation}\} > \{\textit{Payment Traceability}\}\end{array} \quad (2)$$

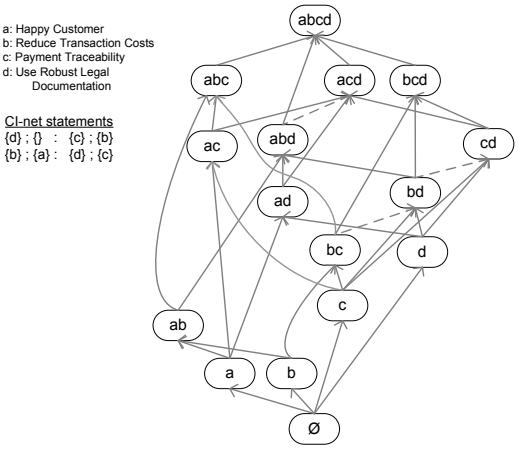

Fig. 2. Induced preference graph for CI-net statements 1 and 2

Figure 2 shows the IPG corresponding to the preferences expressed by these CI-net statements for the illustrative example in Section 2. Each directed edge in the graph represents a "flip" from a less-preferred set of NFPs to a more-preferred set. Solid edges (e.g., from $\{c\}$ to $\{bc\}$) indicate *monotonicity flips*; here, the set $\{$*Reduced Operational Costs, Payment Traceability*$\}$ is preferred because it has one more NFP than the set $\{$*Payment Traceability*$\}$. Dashed edges (e.g., from $\{bc\}$ to $\{bd\}$) indicate *importance flips*, which are induced by a CI-net statement (in this case, by statement 2 above). Figure 2 shows that the set of all NFPs is most preferred, while the empty set (no NFPs satisfied) is least preferred.

3.3 Service Representations and Composition

Labeled transition systems (LTS) [7] represent the low-level behaviors of services in our system. An LTS is a digraph where nodes model states and edges represent transitions. It is given by a tuple: $(S, s_0, \Delta, I, O, AP, L)$, where S is the set of states, $s_0 \in S$ is the start state, $\Delta \subseteq S \times (I \cup O) \times S$ is the set of transitions where each transition is labeled with an input action $\in I$ or an output action $\in O$, AP is the set of atomic propositions, and L is a labeling function which maps each state $\in S$ to a set of propositions $\subseteq AP$. The labeling function describes configurations or states of the LTS. We use the notation $s \xrightarrow{!a} s'$ (resp. $s \xrightarrow{?a} s'$) to denote output (resp. input) action a when the system moves from s to s'.

Figure 3 illustrates the LTSs of available services that can be used to realize the composite service discussed in Section 2. With $AP = \{t_0, t_1, t_2, t_3, t_4\}$ and $S = \{s_0, s_1, s_2, s_3, s_4\}$, the LTS for the *Vehicle* service moves from the start state s_0 to state s_1 after the action `Location`: $s_0 \xrightarrow{\texttt{Location}} s_1$. The state s_1 moves to state s_2 when the service outputs the `Problem` description. This is followed by the input `MechCntct`, which contains the contact information for the mechanic,

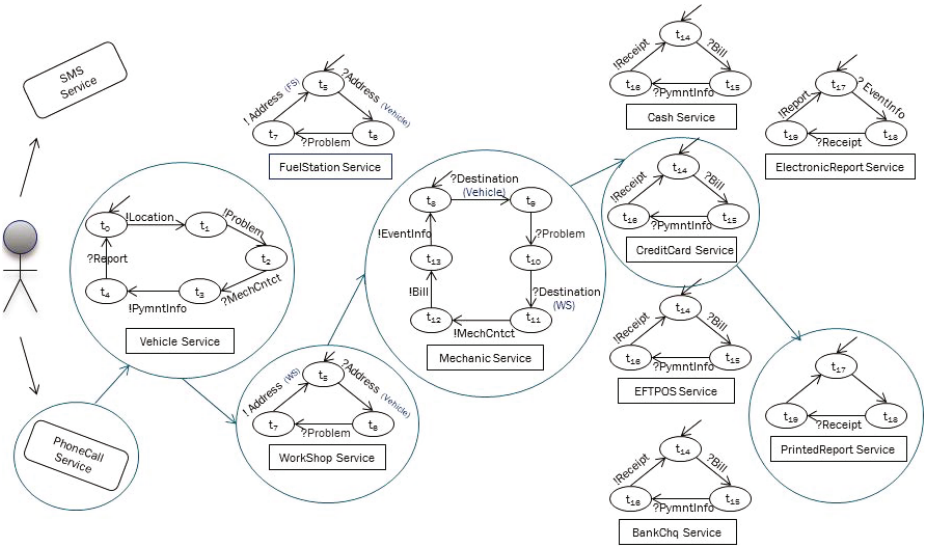

Fig. 3. *HelpMeOut* services. Circled services constitute the final composition.

and then by the output `PymntInfo`. Finally, the system moves back from state s_4 to state s_0 with the output of a `Report` of the incident.

Formally, given $\text{LTS}_i = (S_i, s_{0i}, \Delta_i, I_i, O_i, AP_i, L_i)$ where $i \in \{1, 2\}$ and $AP_1 \cap AP_2 = \emptyset$, the synchronous parallel composition $\text{LTS}_1 \times \text{LTS}_2$ is defined as $(S, s_0, \Delta, \tau, AP, L)$, such that $S \subseteq S_1 \times S_2$, $s_0 = (s_{01}, s_{02})$, $AP = AP_1 \cup AP_2$, $L((s_1, s_2)) = L(s_1) \cup L(s_2)$, and $\Delta \subseteq S \times [\tau]S$: $(s_1 s_2) \xrightarrow{\tau} (s_1' s_2') \Leftarrow s_1 \xrightarrow{!a} s_1' \wedge s_2 \xrightarrow{?a} s_2'$. In other words, a parallel composition of LTSs representing services describes all possible behaviors exhibited by the services via exchange of messages (output from one is consumed by input to another).

3.4 Behavioral Constraints

We use an expressive temporal logic named Computation Tree Logic (CTL) [7] to describe the behavioral constraints. A CTL formula φ is described over a set of atomic propositions AP as follows:

$$\varphi \rightarrow AP \mid \texttt{true} \mid \neg\varphi \mid \varphi \wedge \varphi \mid \texttt{E}(\varphi \texttt{U} \varphi) \mid \texttt{AF}\varphi$$

The semantics of a CTL formula, denoted by $[\![\varphi]\!]$, is given in terms of the sets of states where the formula is satisfied. AP is satisfied in all states which are labeled with the propositions in AP, `true` is satisfied in all states, $\neg\varphi$ is satisfied in states which do not satisfy φ, and $\varphi_1 \wedge \varphi_2$ is satisfied in states which satisfy both φ_1 and φ_2. $\texttt{E}(\varphi_1 \texttt{U} \varphi_2)$ is satisfied in states from which there exists a path to a state satisfying φ_2 along which φ_1 is satisfied in all states. Finally, $\texttt{AF}\varphi$ is satisfied in states from which all paths eventually end in a state satisfying φ.

Example 2. Recall the behavioral constraint specified in Section 2, which stated that if either the *EFTPOS* or the *Cash* service is used for payment *On the Spot*, then a printed report should be sent instead of an electronic report. This can be expressed by the CTL statement $\texttt{AG}((\mathit{EFTPOS} \vee \mathit{Cash}) \Rightarrow \texttt{AX}(\mathit{PrintedReport}))$.

3.5 Data Mismatches

A data mismatch occurs when the input and output actions of two services that can potentially communicate do not match. Data mismatches can be classified as systemic, syntactic, structural, and semantic [15]. Systemic level mismatches are no longer a problem due to standardized network protocols like IP, TCP, and UDP. Syntactic mismatches are automatically resolved by using a standard service description language such as WSDL (Web Service Description Language [4]).

A semantic mismatch occurs when communicating services refer to the same piece of information with different names. For example, in Figure 3 the vehicle's location is sent via an output `Location`, while *FuelStation* and *Mechanic* services expect to consume the location information via input actions `Address` and `Destination` respectively. We address this problem using a data dictionary [1]. The dictionary elements — expressing distinct concepts — are grouped as sets of synonyms, resulting in a collection of meaningfully linked words and concepts.

A structural mismatch occurs when the data received by a service is found in other-than-expected order, style or shape. Differences in number or order of XML tags of interacting services are examples of structural mismatches. We utilize the graph-theoretic solution introduced in [1] to address this problem.

4 Service Composition Framework

Our service composition framework takes as input the entire set of functional requirements, the preferences and trade-offs over non-functional properties, the given behavioral constraints, and a repository of available services. Given these inputs, our framework automatically constructs a composite service that is most preferred (optimal) with respect to the users' non-functional property preferences and that satisfies the functional requirements and all behavioral constraints.

4.1 Specifying the Service Composition Problem

The inputs to the service composition framework are:

Θ A goal model, which shows a Boolean (AND/OR) combination of functional requirements θ and their impact on non-functional properties.
P A preference relation specified using a CI-net, which forms a partial order over the powerset of all NFPs under consideration.
Ψ A set of Computation Tree Logic (CTL) statements ψ that formally describe the behavioral constraints as temporal properties of the composition.
R A repository of services, which are each specified in a standard description language (e.g., WSDL) from which a corresponding LTS has been extracted.

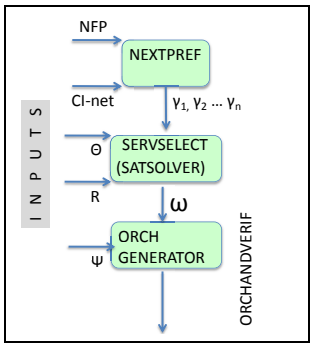

Fig. 4. Overview of framework

The Composition Problem. The objective is to solve the following problem: Does there exist a composition C of services $\in R$ such that

$$\begin{array}{l}[C \text{ satisfies } \Theta \wedge \Psi] \text{ and} \\ \forall \text{ composition } C' \text{ of services} \in R : [C' \text{ satisfies } \Theta \wedge \Psi \Rightarrow C' \not\succ C]\end{array} \tag{3}$$

In other words, our objective is to identify a composition C such that (1) C satisfies all functional requirements and behavioral constraints and (2) no other composition C' that satisfies these requirements/constraints is preferred to C.

4.2 Selecting, Creating, and Verifying a Composition

Figure 4 illustrates the proposed framework for addressing this problem. The **first module** NextPref uses the non-functional properties `NFP` and the CI-net statements describing the preferences and trade-offs over them to compute an ordered sequence $\gamma_1, \gamma_2, \ldots, \gamma_n$. Each γ_i in the sequence represents a subset of `NFP` where $\gamma_{i+1} \not\succ \gamma_i$ with respect to the CI-net statements. In other words, the sequence of γ_is forms a total order consistent with the partial order of the induced preference graph. Based on the techniques proposed in [13], this module represents the induced preference graph (see Figure 2) as an input model of a standard model checker (specifically, NuSMV [6]) and identifies sequence $\gamma_1, \gamma_2, \ldots, \gamma_n$ by verifying carefully selected temporal properties of the IPG.

Example 3. Consider the IPG in Figure 2. To conserve space, let $a =$ *Happier User*, $b =$ *Reduced Operational Costs*, $c =$ *Payment Traceability*, and $d =$ *Robust Documentation*. Clearly $\gamma_1 = \{a, b, c, d\}$ is the most preferred set of NFPs. Next, consider all sets of NFPs with edges pointing to $\{a, b, c, d\}$. Figure 2 contains no edges between three of these sets, which means that none of them are strictly preferred to each other; however, the graph does contain an edge from $\{a, b, d\}$ to $\{a, c, d\}$, which is induced by CI-net statement 1. Therefore, we assign $\gamma_2 = \{a, b, c\}$, $\gamma_3 = \{b, c, d\}$, and $\gamma_4 = \{a, c, d\}$ (although these sets could be in any order). We then assign $\gamma_5 = \{a, b, d\}$, as it is strictly less preferred than γ_4 according to Figure 2. This process continues until all sets of NFPs (including the empty set) have been placed into the sequence.

The **second module** is ServSelect. This module takes into account the goal model representing the overall functional requirement Θ and its relationship with the NFPs, the repository R of available services, and the sequence of γ_is in order of preference starting from γ_1. For each γ_i, the module identifies

- the set of services (say X_i^+) that realize functional requirements which have *only* positive impacts on the non-functional properties in γ_i; and
- the set of services (say X_i^-) that realize functional requirements which have *some* negative impacts on the non-functional properties in γ_i.

Example 4. Consider the NFP set γ_4 in Example 3 and the goal model in Figure 1. Based on the dependencies between services that satisfy functional requirements (hexagons in Figure 1) and the NFPs that each service satisfies, ServSelect identifies $X_4^+ = \{$*PhoneCall*, *CreditCard*, *PrintedReport*$\}$ and $X_4^- = \{$*SMSCall*, *Cash*, *BankChq*$\}$. Services in X_4^+ have only positive impacts on the NFPs in γ_4, while services in X_4^- have a negative impact on some NFP in γ_4.

Next, ServSelect solves the satisfaction problem and identifies the set $\mathcal{W}$ of all sets of services Y such that the composition of all services in Y realizes a set of functional requirements which, when satisfied, result in satisfaction of Θ. Note that the presence of OR-nodes in the graph allows Θ to be satisfied in many different ways. Finally, ServSelect verifies $X_i^+ \subseteq Y$ and $X_i^- \cap Y = \emptyset$. Satisfaction of these conditions ensures that Y is the most preferred set of services that satisfy Θ and the non-functional properties in γ_i. If the conditions are not satisfied by any assignment Y, the module considers γ_{i+1} from the sequence of γ_is. This is repeated until a suitable service set Y is obtained. In the worst case, the least-preferred (empty) NFP set γ_n will be used; when this occurs, $X_n^+ = X_n^- = \emptyset$, making the above conditions vacuously true. Therefore, a non-empty set Y will always be obtained.

Example 5. Initially, ServSelect uses the goal model in Figure 1 and the repository of services that includes all services in Figure 3 to identify all possible compositions of available services that may satisfy the overall functional requirement Θ. Recall the sequence of NFP sets identified in Example 3. Observe in Figure 1 that there exists no combination of low-level functionalities that leads to satisfaction of γ_1 (all NFPs), γ_2, or γ_3. Fortunately, the set of services $Y = \{$*PhoneCall*, *Vehicle*, *WorkShop*, *Mechanic*, *CreditCard*, *PrintedReport*$\}$ satisfies the required conditions for γ_4: $X_4^+ \subseteq Y$ and $X_4^- \cap Y = \emptyset$.

The **third module**, OrchAndVerif, takes as input the set $\mathcal{W}$ of sets of services Y from the ServSelect module and the set of behavioral constraints Ψ expressed in CTL. This module verifies whether there exists an orchestration of services $\in Y$ that satisfies Ψ; it also considers data mismatches when composing services (see Section 3.5). The core of the verification technique is a tableau algorithm which takes services in Y and constructs their orchestration in a goal-directed fashion, possibly including interleaving of services; details of the technique are available in [1]. If the verification fails, a different Y is selected

Algorithm 1. Driver Program

```
1: procedure VeriComp(Θ, NFP, P, Ψ, R)
2:     ⟨γ1, γ2, ..., γn⟩ := NextPref(NFP, P)
3:     for i = 1 → n do
4:         W := ServSelect(Θ, γi, R)
5:         for all Y ∈ W do
6:             C := OrchAndVerif(Y, Ψ, R)
7:             if C ≠ ∅ then
8:                 return C
9:     return false
```

from $\mathcal{W}$ and the process is repeated until a suitable Y is identified (*success*) or all elements of $\mathcal{W}$ have been considered (*failure*). Successful termination of the process results in a set of services which (1) satisfies the functional requirements Θ, (2) satisfies all behavioral constraints Ψ, and (3) is most preferred with respect to CI-net preferences over the set of NFPs.

Example 6. Figure 5 presents the successfully generated orchestration of the most preferred set of services (given in Example 5) that fulfills the behavioral constraints (given in Example 2). Recall that the *PhoneCall* service serves as an interface only, so it is omitted from Figure 5 to avoid complexity.

The entire process is presented in the procedure VeriComp in Algorithm 1. Line 2 invokes NextPref. Lines 3–8 iterate over the sequence of γ_is where ServSelect in Line 4 identifies a set $\mathcal{W}$, and OrchAndVerif is iteratively invoked in Lines 5–8 to identify the most preferred orchestration C.

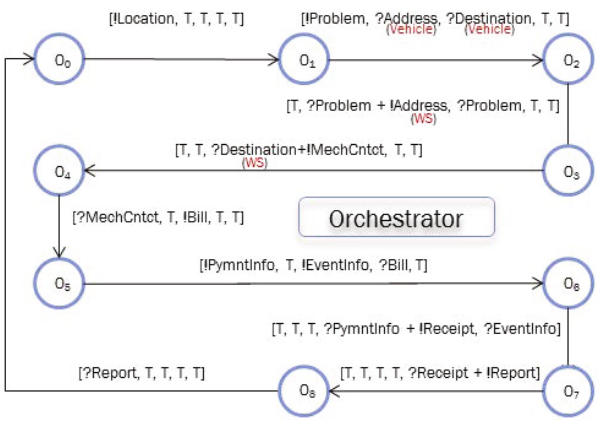

Fig. 5. States and transitions of the synthesized orchestrator. Services are ordered as: [*Vehicle*, *WorkShop*, *Mechanic*, *CreditCard*, *PrintedReport*].

4.3 Theoretical Properties

Theorem 1 (Soundness & Completeness). *Given an AND-OR combination of functional requirements Θ, a set of behavioral constraints Ψ in terms of temporal properties in CTL, a set of non-functional properties* `NFP`*, preferences and trade-offs in CI-nets, and a repository of services R,* VeriComp *returns a composition C if and only if C satisfies the condition in (3).*

Proof. The ServSelect module identifies all sets of services Y such that if the services in Y can be composed, the resulting composition satisfies Θ. Further, given Ψ and a set of services Y as input, the OrchAndVerif module returns a composition C if and only if the services in Y can be composed in a way that satisfies all behavioral constraints in Ψ; this follows directly from results in [1].

It remains to prove that for any composition C returned by our framework and for all other compositions C' that satisfy both Θ and Ψ, $C' \not\succ C$. Suppose in contradiction that $C' \succ C$, i.e., C' satisfies a more preferred set of NFPs than C, but our framework returns C. Let γ_C and $\gamma_{C'}$ be the sets of NFPs satisfied by C and C' respectively. By Theorem 1 in [13], $\gamma_{C'}$ precedes γ_C in the sequence returned by NextPref; therefore, our framework attempts to compose and verify C' before considering C. Because C' satisfies both Θ and Ψ, OrchAndVerif succeeds in creating and verifying C'. As a result, our framework returns C', contradicting our earlier assumption. □

Complexity. Let n be the number of leaf-level functional requirements in Θ, let k be the maximum number of services in R satisfying any requirement $\theta \in \Theta$, and let p be the number of NFPs considered. Algorithm 1 (VeriComp) iterates up to 2^p times over the outer loop (lines 3–8), once per subset of `NFP`. The largest number of possible compositions returned by ServSelect is k^n if all non-leaf nodes in $\mathcal{G}^{\Theta}$ are AND nodes and if the composition has up to n services. The inner loop (lines 5–8) calls OrchAndVerif at most k^n times (once for each possible composition that satisfies the NFP set γ), taking $O(2^n 2^{|\Psi|})$ time per call (where $|\Psi|$ is the number of CTL formulae to be satisfied). The worst-case complexity of our framework, given an AND-OR tree with only leaf and AND nodes, is therefore $O(2^p k^n 2^n 2^{|\Psi|})$. However, we expect k and p to be small in most practical applications. Additionally, each OR node in $\mathcal{G}^{\Theta}$ improves the worst-case complexity by reducing the number of leaf-level requirements to verify.

5 Implementation and Case Studies

We have implemented our framework as a Java-based tool that is based on existing components. The ServSelect module is derived from the goal-model analysis tool in [13], the OrchAndVerif module is based on the composition tool in [1], and the NextPref module is built on the NuSMV [6] model checker.

Table 1 displays results from applying our service composition and verification framework to three case studies adapted from the existing literature. These

Table 1. Results of Applying Our Implementation to Three Case Studies

Case Study	*High-Level Functions*	*Services*	*NFPs*	*CI-net Rules*	*Orchestrator*		*Total Run Time (s)*	*Preference Reasoning Time (s)*	*Orch. and Verif. Time (s)*
					States	*Trans*			
HelpMeOut [1]	9	12	4	2	9	9	2.69	1.44	1.25
Online Bookseller [13]	5	9	4	3	16	16	3.00	1.59	1.41
Multimedia Delivery [14]	8	15	3	2	9	10	2.03	0.84	1.19

results were obtained by running our tool on a machine running Windows 7 Professional (32-bit) with 2 GB of RAM and an Intel Core 2 Duo processor running at 1.83 GHz. Each time shown is the mean of the times observed for 10 runs of that operation on that case study. The preference reasoning and orchestration/verification modules each take time on the order of seconds, while service selection (satisfiability analysis) requires minimal time. This is because the semantics of CI-nets requires exploration of the entire induced preference graph, while the behavioral constraints must be verified with respect to all possible executions of the composite service. These results show the feasibility of our composition framework for real-world applications.

More information on our implementation and on the case studies used in this evaluation is available at `http://fmg.cs.iastate.edu/project-pages/icsoc12/`.

6 Related Work

All service composition frameworks are designed to produce composite services that satisfy users' functional requirements. Some also account for low-level behavioral constraints or non-functional properties, but very few integrate all three in a unified way. The TQoS algorithm [9] provides one such framework. TQoS considers both transactional and QoS properties when composing services, selecting services that have locally optimal QoS for each part of the desired functionality. Additionally, TQoS guarantees by construction that composite services it produces satisfy a standard set of transactional constraints. Our framework goes beyond TQoS in two ways: (1) it more accurately represents users' true preferences between sets of NFPs by using CI-nets instead of a weighted-average method, and (2) it can verify that a composition satisfies *any* behavioral constraint that can be specified using CTL, not just a small fixed set of properties.

The composition method presented in [18] is representative of many techniques that consider both functional and non-functional properties. [18] models the entire problem as an integer linear programming problem, employing simple syntactic matching of inputs and outputs to form the composite service and utilizing quantitative preference valuations for NFPs. In contrast, [17] uses qualitative NFP valuations to select services to compose based on a set of preferences expressed by the user in a different language for qualitative preferences, namely tradeoff-enhanced conditional preference networks (TCP-nets). Our framework's strategy for handling NFP preferences is inspired in part by [17].

In [2], ter Beek et al. focus on verification of functional requirements in a scenario similar to our example in Section 2. The service-oriented architecture in [2] is modeled as a set of state machines illustrated as UML-like diagrams. Temporal constraints representing functional requirements and behavioral constraints are specified in the temporal logic UCTL (an extension of CTL) and verified over the diagrams using an on-the-fly model checking tool. Though [2] does not consider NFPs, it shows that model checking is feasible in an industrial-scale service-oriented computing scenario. Our framework employs model checking for both verifying behavioral constraints and reasoning over users' NFP preferences to construct a service composition that truly satisfies the users' needs.

The matching of I/O variable names or types, known as semantic or concept-based similarity matching, is typically performed using a data dictionary. Liu et al. [10] used the lexical database WordNet [12] to perform concept-based similarity matching, while we use our own universal dictionary [1] for the same purpose. Another data-related operation, data flow (without handling mismatches), is performed via routing the data among the ports of Web services. An example of this is the ASTRO approach [11], where data flow requirements are collected in a hypergraph called a *data net*, which is then transformed into a State Transition System to become part of a planning domain for composition. Because neither semantic matching nor data routing are complete data solutions due to the complex XML schema associated with Web service data types, we proposed a graph-theoretic solution [1] that bridges these gaps by addressing the problem at the XML schema level; this is incorporated into the framework in this paper.

7 Conclusions and Future Work

We have presented a framework for service composition that takes into account high-level functionalities, low-level behaviors, and non-functional properties in order to identify and create the most preferred service composition that provides the required functionality (if such a composition exists). Our framework makes use of user-friendly representations for specifying functional requirements and non-functional properties, but it also uses model checking to obtain guarantees that a composition will satisfy specified low-level temporal properties. We proved that our composition algorithm is sound, complete, and weakly optimal with respect to the user's non-functional property preferences, and we presented initial results obtained from a prototype implementation of our framework.

The next steps for this work are to refine our current proof-of-concept implementation and empirically compare our tool's performance to similar algorithms such as [2] and [9]. Our future work includes allowing partial satisfaction of NFPs as in [5], automating translation of WSDL service specifications to LTSs, exploring different approaches to dominance testing and different semantics for expressing preferences, and applying our approach for service composition to the related problems of service substitution and adaptation.

References

1. Ali, S.A., Roop, P.S., Warren, I., Bhatti, Z.E.: Unified management of control flow and data mismatches in web service composition. In: Gao, J.Z., Lu, X., Younas, M., Zhu, H. (eds.) SOSE, pp. 93–101. IEEE (2011)
2. ter Beek, M.H., Gnesi, S., Koch, N., Mazzanti, F.: Formal verification of an automotive scenario in service-oriented computing. In: ICSE, pp. 613–622. ACM, New York (2008)
3. Bouveret, S., Endriss, U., Lang, J.: Conditional importance networks: A graphical language for representing ordinal, monotonic preferences over sets of goods. In: International Joint Conference on Artificial Intelligence, pp. 67–72 (2009)
4. Chinnici, R., Moreau, J.J., Ryman, A., Weerawarana, S.: Web services description language version 2.0 part 1: Core language. W3C Recommendation, World Wide Web Consortium (June 2007), `http://www.w3.org/TR/wsdl20/`
5. Chung, L., Nixon, B., Yu, E., Mylopoulos, J.: Non-Functional Requirements in Software Engineering. Kluwer Academic (2000)
6. Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: NuSMV 2: An OpenSource Tool for Symbolic Model Checking. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, pp. 359–364. Springer, Heidelberg (2002)
7. Clarke, E., Grumberg, O., Peled, D.: Model Checking. MIT Press (January 2000)
8. Erl, T.: SOA: Principles of Service Design. Prentice Hall (2008)
9. Haddad, J.E., Manouvrier, M., Rukoz, M.: TQoS: Transactional and QoS-aware selection algorithm for automatic web service composition. IEEE T. Services Computing 3(1), 73–85 (2010)
10. Liu, X., Huang, G., Mei, H.: A user-oriented approach to automated service composition. In: ICWS, pp. 773–776. IEEE Computer Society (2008)
11. Marconi, A., Pistore, M.: Synthesis and Composition of Web Services. In: Bernardo, M., Padovani, L., Zavattaro, G. (eds.) SFM 2009. LNCS, vol. 5569, pp. 89–157. Springer, Heidelberg (2009)
12. Miller, G.A.: WordNet: A lexical database for English. Communications of the ACM 38(11), 39–41 (1995)
13. Oster, Z.J., Santhanam, G.R., Basu, S.: Automating analysis of qualitative preferences in goal-oriented requirements engineering. In: Alexander, P., Pasareanu, C.S., Hosking, J.G. (eds.) ASE, pp. 448–451. IEEE (2011)
14. Oster, Z.J., Santhanam, G.R., Basu, S.: Identifying optimal composite services by decomposing the service composition problem. In: ICWS, pp. 267–274. IEEE Computer Society (2011)
15. Ouksel, A.M., Sheth, A.: Semantic interoperability in global information systems. SIGMOD Rec. 28, 5–12 (1999)
16. Pessoa, R.M., da Silva, E.G., van Sinderen, M., Quartel, D.A.C., Pires, L.F.: Enterprise interoperability with SOA: a survey of service composition approaches. In: van Sinderen, M., Almeida, J.P.A., Pires, L.F., Steen, M. (eds.) EDOCW, pp. 238–251. IEEE Computer Society (2008)
17. Santhanam, G.R., Basu, S., Honavar, V.G.: `TCP−Compose`* – A TCP-Net Based Algorithm for Efficient Composition of Web Services Using Qualitative Preferences. In: Bouguettaya, A., Krueger, I., Margaria, T. (eds.) ICSOC 2008. LNCS, vol. 5364, pp. 453–467. Springer, Heidelberg (2008)
18. Yoo, J.W., Kumara, S.R.T., Lee, D., Oh, S.C.: A web service composition framework using integer programming with non-functional objectives and constraints. In: CEC/EEE, pp. 347–350. IEEE (2008)