Robust Dynamic Service Composition in Sensor Networks

Sahin Cem Geyik, *Member, IEEE,* Boleslaw K. Szymanski, *Fellow, IEEE,*
and Petros Zerfos, *Member, IEEE,*

Abstract—Service modeling and service composition are software architecture paradigms that have been used extensively in web services where there is an abundance of resources. They mainly capture the idea that advanced functionality can be realized by combining a set of primitive services provided by the system. Many efforts in web services domain focused on detecting the initial composition, which is then followed for the rest of service operation. In sensor networks however, communication among nodes is error-prone and unreliable, while sensor nodes have constrained resources. This dynamic environment requires a continuous adaptation of the composition of a complex service.
In this paper, we first propose a graph-based formulation for modeling sensor services that maps to the operational model of sensor networks and is amenable to analysis. Based on this model, we formulate the process of sensor service composition as a cost-optimization problem and show that it is NP-complete. Two heuristic methods are proposed to solve the composition problem: the top-down and the bottom-up approaches. We discuss centralized and distributed implementations of these methods. Finally, using ns-2 simulations, we evaluate the performance and overhead of our proposed methods.

Index Terms—Service Composition, Sensor Networks, Service Modeling.

1 Introduction

Wireless sensor networks consist of ensembles of low-cost devices that collect raw measurement data from the environment, transform it through a series of operators into more meaningful aggregate values, and relay these values (possibly over multiple hops) to base stations, for collection and further processing by end-users. Due to limited communication bandwidth, node processing and energy resources, sensor network applications are distributed over a collection of nodes. Each node typically provides a basic functionality for operating on the monitored data, while the network of sensor nodes collectively provides a composite service (i.e. a service that is formed through a suitable combination of basic functionalities [1]) to the end-user.

As an example of a composite service, consider the tracking and object identification application of Figure 1: audio measurements are collected from three acoustic sensors and are used to localize the source of the sound. The localization information is then transmitted to a camera sensor that identifies the type of the object that produces the sound. The camera is further used for tracking the object as it roams in the monitored field. In this example, one can readily identify the primitive functionalities that are collectively used to provide the more sophisticated tracking and object identification service.

Service composition, i.e. the process through which composite services are produced by combining several primitive ones [2], has been the subject of extensive study in the context of web services [1]. However, the unique characteristics of sensor networks render techniques that were devised for web service composition inadequate. Unlike the web environment where service provider availability and ample communication bandwidth are typically assured, sensor networks are highly dynamic as nodes often fail or become disconnected and wireless communication capacity is limited. Thus, web service composition approaches (e.g. [3]-[8]) are susceptible to single-point-of-failure and inefficient use of precious wireless communication bandwidth.

Additionally, the service paradigm exhibits qualitative differences in the web and sensor network domains: in the web, service consumers are typically concerned with finding service instances from a plurality of web providers that can accomplish a given, abstract task or functionality [9], [10]. In sensor network deployments, the service paradigm is primarily concerned with the assembly of data transformation pipelines over data flows. In Section 2, we discuss important implications that this distinction in the service model have on the optimization of the service composition process.

Early programming frameworks [11], [12] proposed for the development of sensor services recognized the need for a component-based design that com-

- *Sahin Cem Geyik and Boleslaw K. Szymanski are with the Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY, 12180. E-mail: sahincem2001@yahoo.com, szymansk@cs.rpi.edu*
- *Petros Zerfos is with IBM T.J. Watson Research Center, Hawthorne, NY, 10532. E-mail: pzerfos @us.ibm.com*
- *The authors would like to thank Dinesh Verma from IBM T.J. Watson Research Labs, and Christopher Gibson from IBM U.K. Research Labs, for the discussions through the preparation of this manuscript. This research was sponsored by US Army Research laboratory and the UK Ministry of Defence and was accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the authors, and should not be interpreted as representing the official policies, either expressed or implied, of the US Army Research Laboratory, the U.S. Government, the UK Ministry of Defense, or the UK Government. The US and UK Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.*

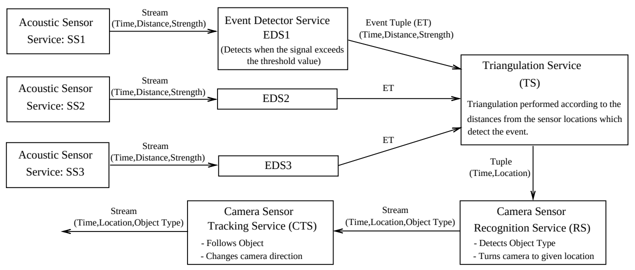

Acoustic Sensor Service: SS1
Stream (Time,Distance,Strength)
Event Detector Service EDS1
(Detects when the signal exceeds the threshold value)
Event Tuple (ET) (Time,Distance,Strength)
Acoustic Sensor Service: SS2
Stream (Time,Distance,Strength)
EDS2
ET
Triangulation Service (TS)
Triangulation performed according to the distances from the sensor locations which detect the event.
Acoustic Sensor Service: SS3
Stream (Time,Distance,Strength)
EDS3
ET
Tuple (Time,Location)
Stream (Time,Location,Object Type)
Camera Sensor Tracking Service (CTS)
- Follows Object
- Changes camera direction
Stream (Time,Location,Object Type)
Camera Sensor Recognition Service (RS)
- Detects Object Type
- Turns camera to given location

Fig. 1. A Composite Service Example

partmentalizes (at the source-code level) the transformational steps that network collected data must undertake. Recent advances in sensor network programming further extend this concept by proposing the use of a high-level language such as Haskell [13], WaveScript [14] or Prolog [15] to describe the interconnection of application components, each of which is implemented in a lower-level, device-specific language. However, such programming models are also not robust enough for the dynamics of sensor network environments, neither do they make efficient use of the limited network resources. There is no provisioning for flexible re-engineering of the sensor application at *runtime* should the nodes that provide the services fail or become disconnected and the service model is not adapted to the ever-changing processing and energy resources of the nodes.

The objective of the work presented herein is to introduce a modeling and composition framework for sensor services that is robust to sensor node and communication failures and efficiently use the underlying resources[1].

In this service-oriented approach, sensor network applications are represented and viewed as a collection of component services assembled in a data flow graph that describes the composite service. Each component service provides basic operators for transforming the data, has typed inputs and outputs, and generates metadata that provides meta-information on the values that are being transformed, as well as on the runtime properties of the sensor nodes that implement the service. Such runtime information may include the cost of processing data at each node and transferring the results between nodes in the network.

The graph-based modeling of sensor services along with the cost information are used to formulate the process of dynamic sensor service composition as a cost-optimization problem, which we further prove to be NP-complete. We devised two heuristic methods to solve this problem, which differ on how the composition process proceeds: the top-down approach starts with the high-level specification of the desired composite service and proceeds in multiple steps of refinement by identifying the primitive component services that can be used to provide its inputs. In the bottom-up approach, the service graph is topologically sorted, and each service waits for its candidate input provider services to decide on their own compositions, then chooses a subset of them to satisfy its own inputs. The top-down approach is motivated by the need to provide cost-efficient service composition graphs due to the limited resources in the network. The bottom-up approach is devised to support robustness in the face of sensor node and communication failures. Centralized and distributed implementations were evaluated for both approaches. In summary, this paper makes the following contributions:

- a modeling framework for sensor services that follows a data flow graph formulation, which is amenable to analysis,
- a formulation of the dynamic sensor service composition process as a cost-optimization problem, which is shown to be NP-complete,
- two algorithms that use heuristics for solving the dynamic service composition problem, along with an analysis of their complexity, and
- evaluation of the algorithms using ns-2 simulations to measure their costs and robustness to node failures.

The rest of the paper is organized as follows. Section 2 outlines a model of sensor services and formulates the composition process in sensor networks. Section 3 describes our two approaches to sensor service composition, namely the top-down and the bottom-up, along with their centralized and distributed implementations. Simulation results are provided in Section 4. Section 5 discusses related work on modeling and composition of web and sensor services. The paper concludes in Section 6.

2 Sensor Service Modeling and Composition

In this section, we first motivate the need for a new modeling and composition framework for sensor network services. Modeling of services utilizing inputs, outputs, and semantic properties, as well as the service composition problem have been examined thoroughly for web services (a detailed description is given in [17]) which have many commonalities with the model we utilize within the sensor networks domain, but with a significant difference in the reliability of the environments in which the services are deployed. For sake of completeness, the later parts of this section present the sensor service model and describe a formulation of the composition process within that model as a cost-optimization problem, which we further show to be NP-complete.

2.1 Motivation

Service modeling and composition have been extensively studied for web services and business processes over a number of years [1], [18]. A number

1. The initial exposition and results of our approach appeared in our previous paper [16].

of standards [19]-[21] have been adopted and widely used in real-world deployments that developed languages and tools for describing web services, enabling automatic discovery, composition, enactment, and monitoring of web services. However, in the sensor network domain, both the unique challenges of the operating environment as well as the data-driven approach of communication call for a rethinking of the services paradigm.

To begin, the limited resources of sensor nodes (caused by energy depletion from batteries, constrained wireless communication bandwidth, low processing capabilities, etc.) make heavyweight web service modeling languages, protocols and frameworks such as WSDL [20], BPEL [19], and SOAP [22] inapplicable for sensor networks. For example, while there exists the notion of cost in web services [5], the communication costs incurred through the interaction of the component services, which is expressed either in terms of number of messages exchanged or consumed bandwidth, is often not modeled explicitly, as it is not typically a concern in that domain. Furthermore, as the levels of resources continuously fluctuate in sensor network deployments, the composite service needs to dynamically adapt to these changing conditions.

Secondly, while web services are commonly assumed to be always available as they are provided by always-on servers and robust cloud infrastructures, node failures and communication disruptions in sensor networks are to be expected and do occur frequently. Any service composition approach that depends on a single centralized planner node is prone to such failures, making the process of composing services less robust. Considering only the costs of composition (as in recent efforts [7]) without taking into account the possibility of node failures is not sufficient for providing fault-tolerant composition in these environments. To avoid single-point-of-failures and increase fault-tolerance in the presence of faulty nodes, it is desirable that the service composition method executes in a distributed manner.

Thirdly, in the web services and business processes case, the service model follows a process- (or workflow-) oriented paradigm, whereas sensor applications implement a data-driven model. One important implication of this fundamental difference lies in the way service composition (and the assorted cost-optimization) approach evolves: in the former case, composition involves the binding (assignment) of an abstract service model and its tasks to service component instances [3], [9]. The latter case is concerned with (potentially partial) matching of the input data requirements of a service to another's output data streams, without possessing an explicit composite service model *a priori*. Consequently, in web services, QoS-measures at the local level can be optimized in polynomial time, while for the sensor environment, as we prove later in Section 2.6, optimization at even the local service selection level is NP-hard.

In summary, the dynamically changing (and limited) resources, the frequent node failures, and the data-driven notion of services in the sensor environment call for a service composition approach that is cost-efficient, robust and suitable for data-driven applications. In what follows, we present such modeling and composition framework that possesses these characteristics.

2.2 Modeling Sensor Network Services

A service s_i in a sensor network is defined by the input data that it accepts, the transformation function that it applies to its input, the output data that it produces, as well as metadata that provide additional information that characterizes the service and its outputs:

$$\begin{aligned} s_i = \{&input_i = (input_{i,1}, ..., input_{i,m}), \\ &output_i = (output_{i,1}, ..., output_{i,k}), \\ &f_i(input_i) \rightarrow (output_i), metadata_i(t)\} \ . \end{aligned}$$

Following the above definition, a sensor network, in a service-oriented sense, can be defined as a set of services, abstracted from the sensor nodes and base station(s) that form it:

$$S = \{s_1, s_2, ..., s_n\}.$$

It is apparent that some services implemented in a sensor network may be only *source services*; namely the services that do not receive any input and only produce data. Furthermore, we can also define a *sink service*; a service that does not output anything and only receives input. In an application, the end-user requesting information is usually represented as a sink service[2].

In the service definition given above, metadata is the information on the services characteristics; i.e. it is the information shared between services to give the properties of the data that is produced by the service such as levels of reliability, security, etc. Metadata may also include cost information and certain characteristics of service inputs and outputs, such as energy consumption per output data produced, processing delays, number of other services that make use of its outputs, etc. The metadata of a service depends on time (t), due to the dynamic conditions of the underlying sensor network environment. Each specific service has separate metadata that is transmitted to other services offered by the sensor network. Metadata information is used in our dynamic composition algorithms to find which services are most cost-efficient to use for a given composite service requested by the end-user. More details about the approach to the modeling of

2. It is possible that the output of the end-user service is needed for higher level services, in which case it is not a sink but an intermediate node in this higher level service.

metadata in the service oriented architecture (SOA) for sensor networks, which we also follow herein, are given in [23]. Alternative approaches, e.g. ontologies, for encoding metadata might also be applicable, however their exact application details are beyond the scope of this work.

2.3 Service Graph of a Sensor Network

Service graph of a sensor network, G_S, consists of vertices representing services and directional edges representing possible flow of data between the services. The edge directed from the vertex of service A to the vertex of service B exists if and only if the intersection of the output of A and the input of B is non-empty. That is, informally, A can provide some of the data that it generates (output) for use by the service B through this directional edge. A formal definition of the service graph is given below:

$G_S = \{V, E\}\ \ where,$

$V = \{s_i\}$ (one vertex for each service) and,

$$E \subseteq V\text{x}V,\ where\, e_{i,j} = \begin{cases} 1 & \text{if } (output_i \cap input_j) \neq \emptyset, \\ 0 & \text{otherwise.} \end{cases}$$

Note that, although not stated explicitly in the graph definition, two services can have an edge between them if and only if the metadata of the connected services match, in addition to having compatible input/output. For example, if a service requires input with certain reliability, only the services that can provide this type of data at the requested reliability level can potentially be connected to the service description with a directed edge. While the above definition of the service graph seems to require exact match between input and output fields of two services to build an edge between them, this requirement can be relaxed by using type hierarchies for the data fields, as described next. It is worth noting that in case one service can provide input to another, the actual flow of data is the responsibility of the underlying network and routing layers. However, through the use of metadata, the service model and the composition algorithms are informed of the communication costs involved when such data flow occurs.

Figure 2 shows a simple example of a type hierarchy, regarding the snapshot (image) collected by a camera sensor that monitors a certain area. There are two areas (A and B) and two quality levels (high and low) for a picture. In the graph that describes the type hierarchy, it can be observed that the information type *High Quality Snapshot of Area A* is a *subtype* of *Low Quality Snapshot of Area A*, since it already includes the information that is contained within the latter type. Another example is *High Quality Wider Snapshot of Area A & B* information type, which includes information for both areas A and B, hence it is a subtype of both *High Quality Snapshot of Area A* and *High Quality Snapshot of Area B*. More elaborate type hierarchies can be built for application specific purposes.

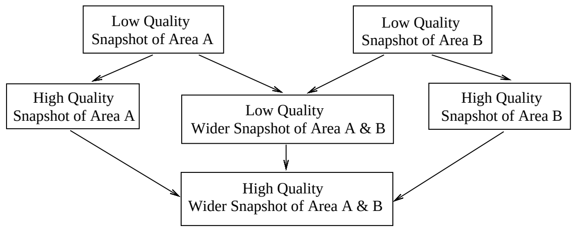

Low Quality Snapshot of Area A
Low Quality Snapshot of Area B
High Quality Snapshot of Area A
Low Quality Wider Snapshot of Area A & B
High Quality Snapshot of Area B
High Quality Wider Snapshot of Area A & B

Fig. 2. A Simple Type Hierarchy

Such a type hierarchy enables two services to be linked in the service graph G_S, even though their outputs and inputs might not match exactly. If a service A's output field is a subtype of a service B's input field, then this means that A can provide the information that B requires and possibly more (what is referred to as "subsumption-based similarity" in [8]). While more complex descriptions of hierarchy using formal logics might be applicable, we strive for simplicity and a light-weight approach in our target environment, hence we did not include them here.

2.4 Cost Formulation of Service Composition in Sensor Networks

There are two basic types of cost related to a composition. The first is the processing cost of each service, i.e. cost incurred by activating an instantiation of a service. The second is the cost of communication between two services exchanging information. This latter cost is interpreted as the edge costs in the service graph defined in the previous section. Note that, while we refer to such values in an abstract way as costs, different types of real costs can be represented. For example, processing cost of a service can be the energy spent by the sensor node that provides this service, the delay that is incurred by this service, etc. The same holds for the communication cost between services. In that case, we are actually using the cost values that are defined by the underlying structure of the network. For example, energy spent by sending information from service A to B includes all the energy spent by the nodes on the route from A to B. Calculation of such underlying costs is related to the mapping from the sensor network topology to the service graph. Therefore, a simplified way of evaluating this cost could be to find the shortest path between two nodes in the sensor network and use this value to compute the cost of communication between the services on these two nodes in the service graph. Note that it is also possible to define a cost vector, which accounts for multiple costs incurred at the same time. Then, compositions can be ordered according to this cost vector using appropriate weights for the cost types. In this paper, we mainly focus on aggregates whose cost is the sum of costs of their components, since energy consumption (which perhaps is the most important factor for sensor networks) is additive. However, the approaches introduced in this paper are capable of utilizing arbitrary cost aggregation schemes, such as delay (maximum delay among services) and reliability (product of component service reliabilities).

2.5 Problem Definition

Service composition requires finding such a set of services $S_C \subset S$ and data flows between those services so that every service in S_C has its inputs provided by at least one other service in S_C. Furthermore, the union of the outputs of services in S_C must satisfy a user-requested functionality Φ, given as a set of output fields required by the end-user satisfying certain properties:

$$\Phi = \{output_{\Phi,1}, ..., output_{\Phi,n}\}.$$

Service composition may be considered as the task of finding a subgraph of the service graph (G_S) defined in the previous section, wherein only a subset of the possible edges (data flows) and vertices (services) is used. Moreover, this problem requires that the cost of the composition is minimized. A formal definition of the problem is as follows: For given $G_S = \{V, E\}$ and Φ, find the minimum cost $V_C \subset V$ and $E_C \subset E$, such that,

$$\Phi \subset \bigcup_{V_i \in V_C} (output\ of\ V_i) \text{ and,}$$

$$\forall V_i \in V_C, (input\ of\ V_i) \subset \bigcup_{V_j\ where\ e_{j,i} \in E_C} (output\ of\ V_j).$$

As it can be seen, an edge $e_{i,j}$ cannot be chosen in the composition scheme unless both V_i (representing service i) and V_j (representing service j) are selected for the composition. According to this problem formulation, the composition process may change dynamically, since the optimal composition is dependent on the network conditions at time t. Services learn the network conditions via the metadata mechanism.

2.6 Service Composition Problem is NP-complete

In a simple version of the above problem formulation of sensor service composition, we have a set of source services, which only produce output, a user-request Φ (a sink service) and we wish to find a subset of these source services so that their output fields will satisfy the input fields of the user-request, while keeping the service cost below a certain level. For purposes of illustration, we assume that this cost is additive, such as the total energy consumed by the services that are used. The service composition problem is more general than the above formulation, but even this restricted version is NP-complete by a simple polynomial transformation from the set cover problem. The set cover problem accepts a set, and a set of subsets of this set with a cost assigned to each subset. The problem is to find a subgroup of these subsets so that the original set is covered and cost is below a given value. As is well-known [24], the set cover problem is NP-complete.

Theorem 1. *Service composition is NP-complete.*

Proof: We will transform set cover problem to service composition. For any set cover instance, let S_{cov} denote the set to be covered and for each $i \leq 2^{|S_{cov}|}$, let Sub_i denote a subset of S_{cov}. For each Sub_i, we are given the cost c_i and the problem is to find a cover S_{cov} with cost smaller than c_{req}. The required transformation is as follows:

- Transform S_{cov} to be Φ, the user-request with the required input as the set to be covered,
- Transform each Sub_i into a service with no input, and output being the same as Sub_i, cost of running this service is c_i.

After such a transformation, a procedure finding the composition with the cost lower than c_{req} will also solve the set cover problem. Therefore the service composition problem is NP-hard. Next we will prove that it is in NP.

Given a composition solution (chosen services and data flow), it takes linear time with input to see that the composition satisfies all the input requirements of the services activated in it and the end-user requirements. The complexity of checking the cost of the solution is $O(|V| + |E|)$, since each edge in the solution will only be checked once and each vertex will only be traversed once. Indeed, each edge may incur a cost of transmission and each service incurs a cost of processing. Since we can check the given solution to a composition problem in polynomial time, the service composition problem is in NP.

By showing that the service composition problem is NP-hard and in NP, we prove that it is NP-complete. Note that the decision version of the service composition problem, which decides if there is a composition below a cost c_{req} is NP-complete; the optimization problem, which calculates the least cost composition, is NP-hard, similar to the set cover problem. □

3 Approaches for Service Composition in Sensor Networks

The algorithms that we present in this section aim at achieving cost optimization across the sensor network, while reacting to changing network conditions by recomposing the service. The two proposed approaches are top-down (similar to *backward chaining*), that proceeds with composition down the service hierarchy, and bottom-up (similar to *forward chaining*) that proceeds in the opposite direction. To ensure that the heuristics terminate, we consider only service graphs that are *acyclic* and *directed*. We leave the formulation of service composition costs on more general graphs to future work.

3.1 Top-down Approach

The top-down algorithm starts when the user-request (which is represented by a *sink* service) is received. It first finds a set of services that satisfy its inputs and minimize the local cost, which is the sum of the cost of services chosen and communication costs between those services and user-request. Later, the chosen services choose their input providers and so on. A key requirement in this scheme is that the services at the same level should be composed one after another. It is easy to see that this approach is indeed a breadth-first traversal in the service graph, G_S, which provides us with the ability to identify which services are already used for composition. However, this breadth-first traversal requirement also makes the top-down approach difficult to implement in a distributed way, since it requires synchronization among sensor nodes.

At each level, a set of services is chosen such that among all sets that can supply the inputs to the service under consideration, the selected set is the least expensive. To make such choice, we employ a heuristic initially proposed for the set cover problem [25], which selects the service that adds the smallest cost per each input covered. *Critical services*, which are those that exclusively provide certain input fields of a service, are also selected. Since these have to be included in any feasible set of services, they are chosen first, in case they also cover additional inputs that would have to be provided otherwise by other non-critical services.

A drawback of the top-down approach is that, at any level, it may choose a set of input providers that cannot be further decomposed (their inputs cannot be satisfied), since it makes local decisions without knowledge of available services at the lower levels of the graph. Each service knows only its immediate neighbors in the service graph.

3.2 Bottom-up Approach

The bottom-up approach sorts topologically the directed and acyclic service graph G_S. This means that each service waits for its candidate input providers to decide on which services they will activate, before itself reaching a decision. Algorithm 1 presents a single level algorithm which composes the input that a service requires assuming that its neighboring services have already run the composition algorithm separately and know the set of services they would use for satisfying their respective inputs at a minimum cost. An important point to note in this algorithm is the presence of a $filter$ function that filters the neighbor list of the service under consideration according to the conditions set by the user. Metadata of the possible input providers of a service are considered, and only the neighboring providers that satisfy certain conditions on the metadata (e.g. service reliability, location of measurements, etc.) are included in the composition graph. Algorithm 1 uses a function called *find_comp*

Algorithm 1 Service Composition Algorithm with an Abstract Method for Choosing a Set of Services used for Input

```
method compose_service_inputs(S)
S.composition_cost = 0
input_S = set of inputs of S
N_S = filter(neighbor list of S,condition list)
for each neighbor N_i in N_S do
    input_i = set of outputs of N_i
    input_N[i] = input_S ∩ input_i
    cost_N[i] = N_i.composition_cost
end for
(S.chosen_services , S.composition_cost) =
                    find_comp(input_S,input_N,cost_N,S)
```

(Algorithm 2) that selects the set of services with the smallest cost.

If the bottom-up method is implemented in a distributed manner, it incurs significant communication overhead to transfer complete composition subgraphs (the full list of services used by a possible input provider) among the neighbors of a service. The alternative approach of only transmitting the additive composition cost of the service upstream is followed instead. Once a service S chooses the set of its input providers, only the cost is transmitted further upstream to services that may utilize S's output. Sending the cumulative cost information incurs a lighter traffic load on the system. However, less information is also made available about the composition of a service's possible input providers, which does not facilitate service reusability, and therefore may result in compositions with low cost-efficiency.

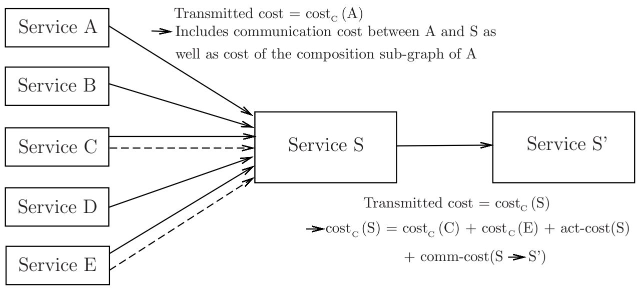

Service A
Service B
Service C
Service D
Service E
Service S
Service S'
Transmitted cost = cost_C (A)
Includes communication cost between A and S as well as cost of the composition sub-graph of A
Transmitted cost = cost_C (S)
cost_C (S) = cost_C (C) + cost_C (E) + act-cost(S) + comm-cost(S → S')

Fig. 3. Sending the Collective Cost Information Upstream

As an example, in Figure 3, service S has five choices that can satisfy its inputs, and chooses two of them, C and E (double arrows) to utilize. S sends its cost information in its $metadata$ to S', as the sum of the collective costs ($cost_C$) of the services it utilizes, its own activation cost ($act\text{-}cost(S)$) and its communication cost to S' ($comm\text{-}cost(S \rightarrow S')$). Such information flows along the edges of the service graph, G_S.

The *find_comp* function required in Algorithm 1 is presented in Algorithm 2, and follows the heuristic

Algorithm 2 Algorithm for Choosing Services to Cover Input Set

```
method find_comp(input,output_sets,cost_set,S)
remaining_in = input
remaining_out = output_sets
chosen_services = $\emptyset$ , comp_cost = 0
while remaining_in != $\emptyset$ do
  if $\exists$ $S_j \in$ remaining_out is a critical service then
    $S_{min}$ = $S_j$
  else
    for each service $S_j$ in remaining_out do
      Find $S_{min}$ where
      $\frac{\text{cost\_set}[S_{min}]+\text{comm\_cost}(S_{min}\rightarrow S)}{|\text{remaining\_in} \cap S_{min}|}$ is smallest
    end for
  end if
  chosen_services += $S_{min}$
  comp_cost += cost_set[$S_{min}$]+comm_cost($S_{min}\rightarrow$S)
  remaining_in -= remaining_in $\cap$ $S_{min}$
  remaining_out -= $S_{min}$
  if remaining_out == $\emptyset$ then
    break
  end if
end while
return (chosen_services , comp_cost)
```

for the set cover. At each step, it attempts to find the neighboring service that has the smallest cost per new input field. The cost is calculated by adding the composition cost of the neighbor node (see Figure 3) and communication cost between this neighbor and the service that is being composed. As in the top-down approach, critical services are always included first due to the input fields that they exclusively provide.

In the evaluation section, we will show that the bottom-up approach achieves lower cost than the top-down approach, and, if there is a satisfying composition solution, it always finds it. The disadvantage of the bottom-up approach is that it requires an acyclic service graph G_S.

3.3 Complexity and Approximation Ratio Analysis

The time complexity for the top-down and bottom-up algorithms can be computed as follows: for the top-down algorithm, each service looks at only its immediate input providers, or neighbors, and each service can have at most $|V - 1|$ of them, where $|V|$ is the total number of services in the system. Since each service makes input checks as well as cost checks, $O(|V|^2)$ operations are performed at each step to choose a set of input providers using the greedy algorithm to minimize cost. Since only a fraction of services could be choosing the best service among the remaining ones, the top-down approach takes $O(|V|^3)$ time to complete.

In the bottom-up algorithm, for each service (Algorithm 1), there is a single call to $find\_comp$ (Algorithm 2) to find its respective set of input providers. This algorithm takes $O(|V|^2)$ time to complete. Since $find\_comp$ is called once for each service, the complexity of the overall bottom-up algorithm is $|V|$ times the complexity of $find\_comp$, i.e. $O(|V|^3)$, which is the same as the top-down approach.

As it was previously shown, sensor service composition problem is NP-complete and we proposed two approaches that use heuristics. We will now briefly discuss their approximation ratio. Feige has proven that the lower bound for approximation of set cover problem is $(1 - o(1))$ lnn [25] unless there are quasi-polynomial algorithms for the problems in NP (n is the size of the set to be covered). This lower bound is achieved by the greedy search that chooses the subset that covers the uncovered elements of the set by selecting the smallest cost per element at each step. This algorithm has a $O(ln$n$)$ approximation ratio and the method $find\_comp$ based on this greedy solution is given in Algorithm 2.

In conclusion, the greedy algorithm finds a composition of cost at most logn$.opt$ at each step of composition, for each service. In the formulation, opt is the cost of the optimal solution and n is the number of input fields that should be satisfied by a service. From this analysis, the overall composition that satisfies an end-user's request can be at most logn$^k.opt$, where k is the furthest distance from any source service in a composition graph to the final output formed by this composition. k can be at most the depth of the acyclic service graph G_S.

3.4 Implementing the Composition Decision Algorithm

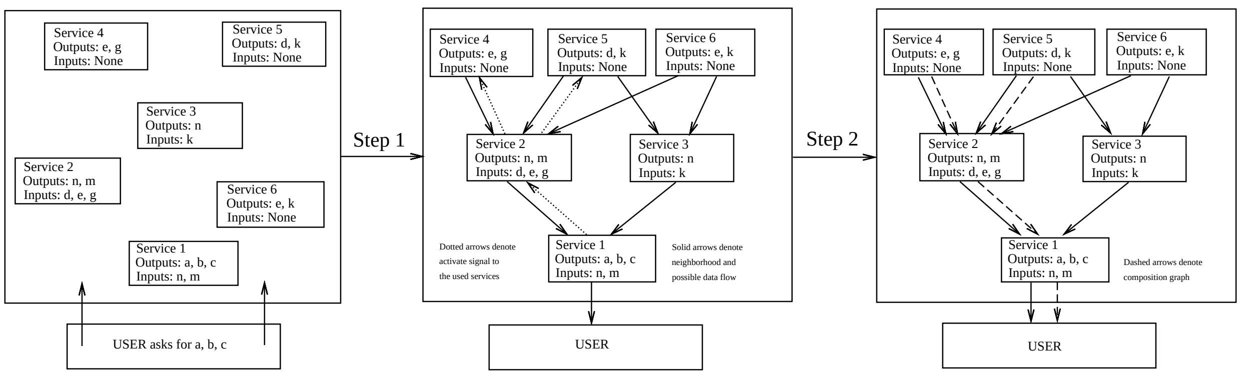
The decision making during composition construction can be either *centralized*, at a central decision node receiving information from all services, or *distributed*, wherein each service separately chooses (locally at the sensor node that they reside on) which services it will use to satisfy its inputs. The details of these two approaches are discussed next.

3.4.1 Centralized Implementation

The centralized approach uses a central decision-making node, where metadata from each service is first collected. Once metadata is received from all services, the decision maker can run either the top-down or bottom-up algorithms and decide on the services to be activated. Our previous evaluation via numerical experiments on random service graphs [16] shows that, although the bottom-up algorithm creates on average lower cost compositions than top-down approach does, choosing the best of the two at each run is an even better approach. Reference [16] provides more details.

3.4.2 Distributed Implementation

In the distributed implementation, each service decides independently from the others on the services



Service 4 Outputs: e, g Inputs: None
Service 5 Outputs: d, k Inputs: None
Service 3 Outputs: n Inputs: k
Service 2 Outputs: n, m Inputs: d, e, g
Service 6 Outputs: e, k Inputs: None
Service 1 Outputs: a, b, c Inputs: n, m
USER asks for a, b, c
Step 1
Dotted arrows denote activate signal to the used services
Solid arrows denote neighborhood and possible data flow
USER
Step 2
Dashed arrows denote composition graph
USER

Fig. 4. Service Composition Process in the Distributed Algorithm

that it will employ to satisfy its inputs[3]. The advantage of this scheme is its robustness to network faults and the quick reaction to changes in the network conditions. For a composition to change, services do not have to send their information to a centralized decision-making point, which might constitute a single point of failure and bottleneck. Only the bottom-up approach is applicable for a distributed implementation, according to which a service needs and collects information only about its own neighboring services.

When a composition process starts, messages are sent from the end-user requesting service (considered to be at the top level of the service graph) to lower level services. User-requested data has certain properties that are provided at the time of the request. According to these properties, a set of services whose outputs can satisfy the request will be considered potential neighbors of the user-request. This process repeats until this information is disseminated downstream to all the source services, which do not have any inputs. Once this information has reached the source services, the reverse dissemination process occurs, in which, at each stage, the service composition algorithm is executed. Once every service is aware of its smallest composition cost, backward messaging takes place, where certain services are activated. This finalizes the composition graph.

In Figure 4, a distributed composition example is presented. The solid arrows denote the feasible information flow between services in the service graph G_S. The dotted arrows denote the activation decision of services after running the distributed composition algorithm. The dashed arrows denote the final composed graph, which is activated for operation.

3.5 Dynamic Composition

In a sensor network environment, the conditions of services can change fairly frequently. Hence it is important to be able to dynamically change service composition. For this purpose, we rely on $metadata$ information exchange throughout the network.

For the centralized implementation, dynamic composition is very similar to producing the initial one: for each change in the system (e.g. change of a service's cost, or availability), the composition process is re-executed due to the new costs and available service set. The notification of the changes in the network conditions are sent to the centralized decision maker to make such recomposition possible.

In the distributed case, each service will decide on its new input providers based on the updates in its neighbors. When a service updates its own information (e.g. activation/deactivation, change of processing cost), it also notifies the other services that may utilize its output, so that they may change their respective compositions if they choose to do so. Furthermore, if a service (or a connection to a service) is not utilized anymore by the currently activated composition, a signal will be sent to this service to stop itself or one of its links. If the service is not being used by any other services currently in the active composition graph, it shuts down after sending notifications of its status change to every other service it utilizes for its inputs. This way, the current composition can be locally updated when changes occur. If the local (re)composition is not feasible, then the request is propagated upstream in the service composition graph, until a valid composition is achieved. Such a distributed scheme can fall into local minima (as opposed to recomposition performed in a centralized manner), as we further demonstrate in Section 4. As usual, services send periodic "hello" messages to their neighbors to inform them of their availability and employ timeouts to avoid failures that might arise when connectivity is lost because of unexpected network faults.

Finally, an issue with dynamic composition is that of the frequency of updates, which might lead to high composition overhead when some metrics change constantly (e.g. residual energy). Such a situation can be prevented by setting up a composition update period and a service can wait for significant changes

3. Note that distributed composition in this context is different than parallel computation of the composite service graph, which is not considered in this work.

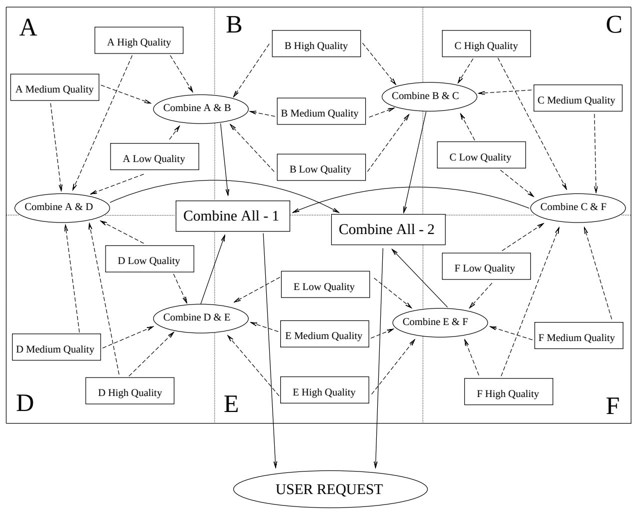


Fig. 5. Simulated Sensor Network Application

in its condition (a threshold) before notifying the central decision maker or the neighboring services about them. These thresholds are application-specific and if needed should be set by the user.

4 Evaluation of Dynamic Composition

To evaluate the dynamic composition capabilities of the distributed and centralized schemes under changing network conditions and node faults, we implemented a composite sensor service application scenario in the ns-2 simulator. The centralized implementation uses the best of bottom-up and top-down approaches, while the distributed implementation uses exclusively the bottom-up approach, which sends only the collective cost upstream. We present the overhead of both approaches, the cost of the compositions they generate over the simulation period, the activation ratio for a feasible composite service and the time it takes for both approaches to react to changing network conditions.

Application scenario Figure 5 illustrates the application scenario that serves as the basis for the simulations. It assumes a monitored field consisting of six areas labeled A to F. In each area, there are three types of source services: low, medium and high quality visual monitoring (e.g. *A High Quality*). There are also intermediate fusion services, e.g. *Combine A & B*, that receive input from two area services and provide an intermediate result whose exact functionality is not important for the purposes of the simulation. We assume that these intermediate services also come in three variants (e.g. *Combine A & B High Quality*), not all of which are shown in the figure, for simplicity. There are also several final fusion services that use the output of the three intermediate services to provide a result for the whole area, e.g. *Combine All - 1*. The user request is submitted through one of the fusion services.

The processing and communication costs have been set according to the size of the outputs produced. For example, the processing cost of the source services were set as 20, 30 and 40 cost units for *Low*, *Medium* and *High Quality* respectively, following the size of the outputs of these services that was set to 4, 8 and 12 size units. We also assume that the *communication cost* per hop and size unit is 5. For example, transmitting the output of a high quality source service between two sensor nodes over a single hop would cost $5 \times 12 = 60$ cost units. The intermediate fusion services that process *High*, *Medium* and *Low Quality* have processing costs of 20, 30 and 40 cost units respectively. The size of their output is set to 6 size units. Lastly, each fusion service has a processing cost of 15 cost units, and an output size of 3.

Implementation in ns-2 The proposed service composition schemes are implemented in ns-2 as an application-layer protocol, independent of the exact underlying routing and MAC layers used for the messaging among nodes. For our simulations, we used *AODV* and *802.11 MAC* as provided by the ns-2 distribution in version 2.34.

Our ns-2 implementation consists of two application layer agents: *ServMeta* and *ServHolder*. ServMeta agent represents a service description (one instance per each distinct service) and includes the service metadata. ServHolder agent is assigned to every node in the network. It is a container agent to which ServMeta agents are assigned. ServHolder agents exchange messages with each other containing information about the services that are assigned to them.

Services are assigned to nodes as follows: each source service is assigned randomly to one of the nodes in its relevant area of monitoring, and there is one instance for each level of quality. Two instances of each intermediate and final fusion service are activated and a single instance of user request is assigned to a sensor node, which never gets deactivated throughout the simulation time. Additionally, for the centralized implementation, we assign a decision maker to one of the nodes that may also get deactivated, to quantify the effect of single-point-of-failure.

After service assignment, the composition protocol for both the centralized and distributed approaches starts with the *service discovery phase*: each node floods metadata (cost, type and size of outputs/inputs) of the services that reside on it to every other node. This way each service discovers what other services exist in the network, and if they are neighbors in the services graph. It also learns the hop-distance between nodes, which is needed to calculate communication cost. Messaging cost for this discovery stage is $O(n|S|)$, where n is the number of nodes and $|S|$ is the number of services that reside on the sensor network. Following the service discovery phase, the centralized and distributed composition steps are followed as

Parameter	Value
Field Size	120x80
Area Size	40x40
Number of Areas in Field	6
Communication Radius	50
Node Activation Ratio	60%–100%
Node Inactivity Time	160–240 secs (uniformly distributed)
Service Processing Cost Change Frequency	Every 20–400 secs
Service Processing Cost Change Period	1600–2400 secs (uniformly distributed)
Service Processing Cost Change Percentage	+ 0%–40 % (uniformly distributed)
Number of Nodes	30 (5 per area)
Total Simulation Time	20,000 secs

TABLE 1
NS-2 Simulation Parameters

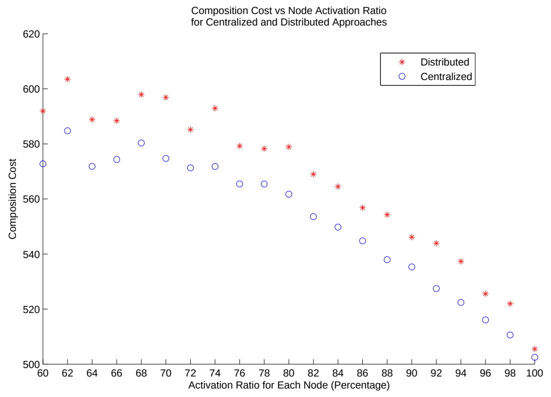


Fig. 6. Comparison of Composition Cost of Centralized and Distributed Approaches for Varying Node Activation Ratios

described in Sections 3.4 and 3.5.

Simulation setup The parameters of the simulation experiments run in ns-2 for the aforementioned application scenario are shown in Table 1. *Node Activation Ratio* represents the expected amount of time a node remains inactive (with the services that run on it being unavailable) compared to the expected time it stays active for the duration of the simulation, which is $\frac{1-activation\_ratio}{activation\_ratio}$. *Node Inactivity Time* denotes the amount of time a node remains inactive, once it changes its status to being so. *Service Processing Cost Change Frequency* denotes the inter-arrival time of events when we increase the cost of a random service in the network by 0%-40% (*Service Processing Cost Change Percentage*). Service cost falls back to its original value after a time period of length between 1600-2400 seconds, uniformly distributed (*Service Processing Cost Change Period*).

4.1 Results

The centralized and distributed composition scheme implementations are evaluated in the erroneous conditions of the sensor network deployments through experiments that vary the activation ratio of each node in the simulated network, while keeping the processing costs of each service (Table 1) constant.

Figure 6 presents the composition costs of the centralized and distributed approaches as a function of the activation ratio. Each value represents the average of 10 runs for each activation ratio and the cost is calculated only for those times during the simulation when a feasible composite service exists. It can be seen that the centralized approach performs better than the distributed one as it generates composition graphs that exhibit lower total processing and communication cost. This is due to two factors: first, the centralized implementation chooses the best of top-down and bottom-up approaches. Second, the recomposition process of the distributed scheme is performed locally, hence it can fall into local optima during the simulation. Centralized composition recomposes the service each time a change in the status of a node takes place. From the figure, one can also observe that there is a decreasing trend in the cost for both approaches as the node activation ratio increases; as more service instances become active concurrently, the probability that a solution is found with a lower cost that would require fewer recompositions increases. It should also be noted that the choice between the best of top-down and bottom-up approaches in our centralized composition scheme gives a better overall performance compared to prior centralized composition schemes [7].

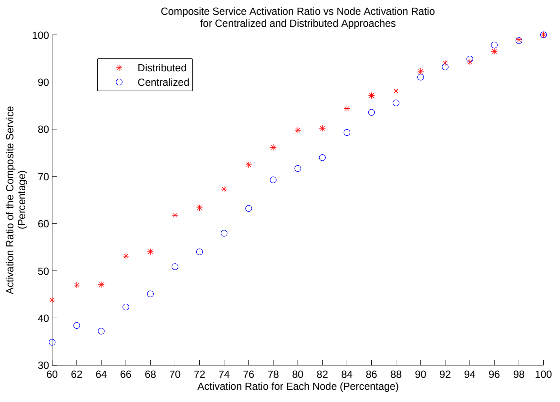


Fig. 7. Comparison of Service Activation Ratio of Centralized and Distributed Approaches for Varying Node Activation Ratios

Figure 7 shows the service activation ratio, defined as the percentage of simulation time when there was a feasible, active composite service. It can be seen that due to the local recomposition process, the distributed implementation is more resilient than the centralized approach, evident by its higher service activation ratio. Furthermore, it is more robust to the single-point-of-failure of the decision making node from which the

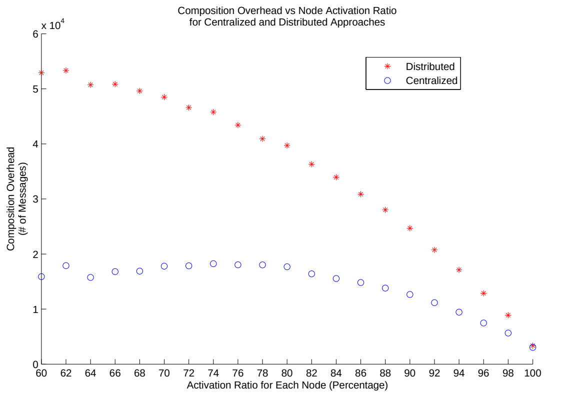


Fig. 8. Comparison of Overhead of Centralized and Distributed Approaches for Varying Node Activation Ratios

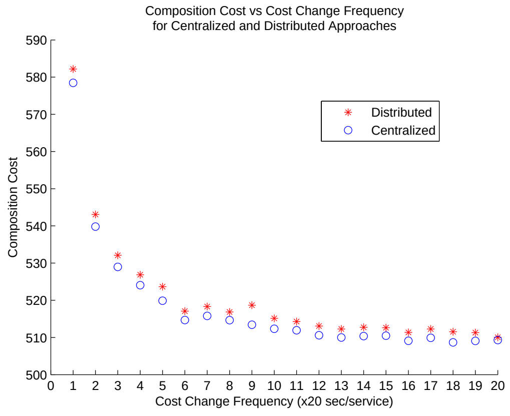


Fig. 9. Comparison of Composition Cost of Centralized and Distributed Approaches for Varying Cost Change Frequencies

centralized approach suffers, which becomes deactivated once such a failure occurs. Any composition scheme, including the logical programming based approaches [15] can perform no better in terms of activation ratio than the centralized version presented herein. Furthermore, the distributed decision making approach for composition that we propose exhibits increased robustness in a highly erroneous environment such as a sensor network.

The comparison regarding messaging overhead, measured as the number of messages that are exchanged between two nodes is shown in Figure 8. The plot also includes the overhead caused by messaging during the *flooding phase*. Clearly, the distributed scheme incurs much higher overhead than the centralized one because changes of a service have to be broadcast to all of its neighbors as opposed to a single centralized decision making node. There exists a trade-off between the activation ratio and the composition cost, complicated by the difference in overheads of the centralized and distributed approaches, making the choice between these two schemes application-specific. For example, an application that can afford low activation ratios but has to be energy-efficient may run the centralized composition scheme, while a mission-critical service that requires high robustness to node failures would be more effectively served by the distributed implementation. Such a trade-off also affects the scalability of the approaches since with many services activated, the overhead is large.

Our second experiment keeps the node activation ratio at 100% to study the effect of processing cost changes for services in the sensor network. As mentioned before, in this work, we use the abstract notion of cost (processing and communication), which can be interpreted as quality of service metrics or actual costs such as energy spent. The results of this experiment are given in Figure 9 and Table 2.

In Figure 9, the distributed and the centralized composition implementations are compared in terms of the cost of the compositions obtained, where the frequency of changes of service cost varies every 20-400 seconds *on average* in steps of 20 secs. The mean value for the frequency is denoted on the x-axis of Figure 9. The resulting composition costs denoted on the y-axis are averaged over 10 runs, each lasting for 20000 secs of simulated time. As expected, Figure 9 shows that more frequent service cost changes (less time between cost changes in the figure) brings higher cost compositions for both approaches. Centralized implementation again performs slightly better, and the reasons for this are similar to the aforementioned experiment with changing activation ratios.

	Distributed	Centralized
First Reaction (sec)	0.0751	0.0080
Last Reaction (sec)	0.3198	0.8036

TABLE 2
Comparison of Reaction Time of Centralized and Distributed Approaches for a Service Processing Cost Change in the Sensor Network

To demonstrate the agility of our approaches in adapting to the dynamic conditions of the sensor networks, we present in Table 2 the average reaction times to a change in the processing cost of a service in the sensor network. We denote as "first reaction" the time it takes for an initial reaction (to a service's cost change in the system) such as recomposition, addition or removal of a service to/from the current composite service graph. The centralized approach performs better for this metric; when there is change in the processing cost of a service, only a single message is sent to the centralized decision making node, in contrast to the distributed case where all services that may utilize this service's outputs have to receive a notification message. The latter leads to increased messaging overhead and latency. The "last reaction" represents the time when the last action (i.e. recomposition, addition or removal of a service to/from the current composition graph, etc.) is processed by the network. Thus, it indicates the time after which the composition is stable again. For this metric of last reaction, the distributed scheme performs better, due to its ability to recompose locally, making it more suitable for error-prone and unstable environments such as those of sensor networks. This ability to quickly recompose results in an increased robustness of the distributed method.

5 Related Work

Service composition for web services has been the subject of intense study over the past several years [18], [26]. In the following, we discuss the prior work that is most closely related to ours, along with the related field of sensor network programming [27], [28]. In the domain of web services, many works have focused on the subject of *QoS-aware web services composition* [3], [4], [8]-[10], in which an abstract service composition graph is provided to the system, and the problem is to bind actual services to the abstract services on this graph. If the system tries to optimize an aggregate QoS function [29] comprised of different metrics, then this problem turns out to be a variation of the multi-variable assignment problem, hence it is NP-hard.

Mao et al. [10] propose *Automatic Path Creation* service (APC), which is a centralized dynamic web service composition method that considers quality of service (QoS) and network characteristics. This method looks for a shortest path from the end-user to the primitive services, but does not consider the case where a subset of the outputs of a service can be used as input to another service.

One of the main methods utilized to optimize the metrics considered in web service composition is genetic algorithms. In [3], the authors solve the problem of assigning (binding) concrete services to the abstract services in the already given service composition graph according to QoS measures via using the genetic programming approach. When the QoS values deviate, their method applies a recomposition, i.e. rebinding of the services. In [8], the functional metric and semantic quality fit of services is used, as well as QoS optimization to assign services to tasks.

Another method applied in *QoS-aware web services composition* is integer programming. In [4], integer programming is used to assign service instances to the abstract services defined in the task graph. The authors provide negotiation techniques to reach a feasible solution when the constraints set by the user are too limiting. A mixed solution to the same problem is also given in [9], where both local and global optimization is considered to maximize QoS in the composition. The assignment is polynomial time solvable for local optimization for *QoS-aware web services composition*, while in our case it is NP-complete. For global optimization of QoS metrics, [9] also makes use of integer programming.

The problem we focus on here is inherently different from *QoS-aware web services composition*. First, we are not concerned with binding services to tasks whose connections are already given in the predefined task graph. Instead, the user presents a set of outputs that should be provided by the overall system, and the composition graph is generated using the available service instances. As we have shown, it is not always possible to replace services, and in our target domain, even the local optimization of cost is NP-complete. Another difference lies in the composition mechanism. Previous work provides centralized decision making, which can lead to frequent single-point failures in the error-prone sensor networks, while our work proposes a distributed decision making scheme, where each service decides on its input providers locally. As shown in Section 4, the distributed decision making leads to faster recomposition of services and therefore increases activation ratio of the composite service, hence increases robustness.

Another class of related work that is close to ours makes use of logical programming, in which services are described as a set of pre- and post-conditions [5], [7], [30]-[32]. Furthermore, the methods of forward and backward chaining (resembling our bottom-up and top-down methods) are applied to satisfy the conditions set by the user.

[5] proposes *OWLS-Xplan* which is a logical planner that utilizes the service definitions given in OWL-S (Web Ontology Language - Services) language. This way, the conditions can be set by the user and satisfied according to the pre- and post-conditions of the services. Another solution is provided in [7]. Similarly to our bottom-up approach, it utilizes forward chaining and uses an aggregated QoS measure to choose provider services at each point of composition along with pre-/post-conditions, both on the properties and the types of inputs/outputs that match between services. Significant differences from our approach are that in our method, we allow for distributed decision making, which increases robustness in the presence of node failures. Moreover, our centralized approach combines a mix of top-down (similar to backward chaining) and bottom-up (which resembles forward chaining) algorithms, which produces better results. We also show that the local optimization problem is NP-complete and provide an approximate solution. Finally, we address the communication costs and activation ratios based on services availability. These differences also apply to other logic programming-based approaches.

Another example which is given in [30] provides a logical programming based composition of services based on pre-/post-conditions of services, however, it does not take into account the QoS or cost measures. MARIO approach is introduced in [31]. It utilizes tags chosen by the user to provide possible composition schemes. Each tag represents a functional goal and can be interpreted as a query. Type hierarchies are used to connect outputs of a service to compatible inputs of another service in the composition decision process. This work however, does not take into account the changes in the network for performing a re-composition.

The use of the OWL-S language to describe the web services with their inputs and outputs is discussed in [32], [33] and [34]. The first of these references introduces methods for translating services described in OWL-S to SHOP2-compatible descriptions (a logical planner) to compose a user-request. The planner makes use of pre-/post-conditions of described services. However, this method does not allow for any type of distributed decision making. It also considers neither the maximization of QoS, as some previous work did, nor minimization of cost, as we do. The methods in [33] and [34] are user supervised and a set of possible matches for the user functionality needs is presented to the user who selects one or more services for use at each step.

A very similar composition problem based on the matching of multiple outputs and inputs of semantic web services is studied in [35] and [36]. In [35], the authors employ a shortest path algorithm (as opposed to our top-down and bottom-up approaches) on the service graph generated by the match between outputs and inputs of the semantic web services in the system. The authors of [36] utilize a *Semantic Link Matrix* (SLM) of services which is constructed using the similarity between the outputs and inputs of the available web services. A regression-based search on this SLM is proposed for automated web service composition. Both of these methodologies are centralized, and they do not consider activation/deactivation of services (although [36] does mention the updating of SLM with newly introduced or removed services). Naturally, they also do not consider fast reaction to service activation/deactivation which is vital in the domain of sensor networks.

In mobile networks, the authors of [37] propose the use of service equivalence (both semantical and syntactical) to replace services when the connections between nodes are changing rapidly. [37] assumes a pre-defined composition graph for the initial operation, hence the replacement uses other services that can function as well as the current services in the graph. Furthermore, other approaches for service composition, such as Petri-Nets [38] have been proposed in the literature, however they do not take into account any cost or QoS measures.

In sensor networks, few approaches have been proposed for service composition. A noteworthy one is [15] in which the authors provide a method based on logical programming through backward chaining for combining services. They model services as statements whose truth depends on their predicates and they set certain statements true when these predicates are satisfied. These statements are further used by other services as predicates. The method is used for automated inference in sensor networks. Another approach in the sensor networks domain [39] attempts to identify the service composition that is less likely to be invalid in the near future due to nodes going to sleep mode etc. The goal is to minimize the recomposition cost at a later time. In [40], the authors propose components for a network of sensors and actuators from which the complex desired services can be composed. However, the composition process is entirely user-driven.

In [41], the authors propose a dynamic flow control solution, applicable to sensor networks, which uses filters and wires between services. By using filters on the wires (which are logical conditions), the user manually blocks data flow whenever such blocking is needed for the functionality desired in the current network conditions. This system still requires user interaction. Another programming framework, *EnviroSuite* [42], abstracts external environmental elements into objects, hence simplifies sensor network implementations. EnviroSuite is appropriate for implementing the service modeling we propose herein, however it does not include automated composition, which, to the best of our knowledge, is novel in the sensor domain.

A model similar to our service graph can be found in [43], which proposes abstract task graphs that consist of abstract tasks and abstract channels. These are mapped to services (nodes) and possible connections (edges) in the service graph, respectively. However, this paper does not address automatic composition construction or cost measures. In [44], the authors present MiLAN, which is a middleware for sensor applications. MiLAN receives the application requirements for the needed information and chooses a set of sensors that can provide this information according to certain quality of service requirements. However, MiLAN does not provide composition of services in which outputs of services are combined to provide inputs of other services.

6 Conclusions and Future Work

In this paper, we have described a novel method of service modeling and dynamic composition, which is appropriate for the unreliable and unstable sensor network environment. Based on the sensor service model, the problem of sensor service composition has been formalized and was proven to be NP-complete. Two heuristics-based algorithms for dynamic composition were then described, namely the top-down and the bottom-up approaches, based on the direction of flow of information during the composition process. Centralized and distributed implementations along with their advantages and disadvantages were also discussed. The dynamic composition evaluations were further performed through simulations.

Our future work includes the implementation of the algorithms on an actual sensor network environment, to further evaluate the approaches under realistic conditions. Furthermore, we are planning to follow a game theoretic approach to choosing services with minimal cost. In such a scheme, each sensor node will aim to lower its load by utilizing services implemented on other sensors rather than itself. We believe that such a scheme will lead to lower cost solutions, after the initial composition is done with our current algorithms.

REFERENCES

[1] S. Dustdar and W. Schreiner, "A survey of web services composition," *J. Web and Grid Services*, vol. 1, pp. 1−30, Aug. 2005.

[2] R. Hull and J. Su, "Tools for composite web services: A short overview," *ACM SIGMOD Rec.*, vol. 34, pp. 86−95, Jun. 2005.

[3] G. Canfora, M. Di Penta, R. Esposito, and M. L. Villani, "A framework for QoS-aware binding and re-binding of composite web services," *J. Syst. and Softw.*, vol. 81, pp. 1754−1769, Oct. 2008.

[4] D. Ardagna and B. Pernici, "Adaptive service composition in flexible processes," *IEEE Trans. Softw. Eng.*, vol. 33, pp. 369−384, Jun. 2007.

[5] M. Klusch, A. Gerber, and M. Schmidt, "Semantic web service composition planning with OWLS-Xplan," in *Proc. AAAI Fall Symp. Semantic Web and Agents*, 2005, pp. 55−62.

[6] G. Canfora, M. Di Penta, R. Esposito, and M. L. Villani, "An approach for QoS-aware service composition based on genetic algorithms," in *Proc. GECCO*, 2005, pp. 1069−1075.

[7] P. Bartalos, "Effective automatic dynamic semantic web service composition," *Inf. Sci. and Technol. Bulletin ACM Slovakia*, vol. 3, pp. 61−72, Mar. 2011.

[8] F. Lécué and N. Mehandjiev, "Seeking quality of web service composition in a semantic dimension," *IEEE Trans. Knowl. Data Eng.*, vol. 23, pp. 942−959, Jun. 2011.

[9] L. Zeng, B. Benatallah, A.H.H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "Qos-aware middleware for web services composition," *IEEE Trans. Softw. Eng.*, vol. 30, pp. 311−327, May 2004.

[10] Z. M. Mao, R. H. Katz, and E. A. Brewer, "Fault-tolerant, scalable, wide-area internet service composition," Univ. California, Berkeley, Tech. Rep. UCB/CSD−01−1129, 2001.

[11] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler, "The nesC language: A holistic approach to networked embedded systems," in *Proc. ACM SIGPLAN*, 2003, pp. 1−11.

[12] B. Greenstein, E. Kohler, and D. Estrin, "A sensor network application construction kit (SNACK)," in *Proc. ACM SenSys*, 2004, pp. 69−80.

[13] G. Mainland, G. Morrisett, and M. Welsh, "Flask: Staged functional programming for sensor networks," in *Proc. ACM SIGPLAN*, 2008, pp. 335−346.

[14] R. Newton, S. Toledo, L. Girod, H. Balakrishnan, and S. Madden, "Wishbone: Profile-based partitioning for sensornet applications," in *Proc. NSDI*, 2009, pp. 395−408.

[15] K. Whitehouse, F. Zhao, and J. Liu, "Semantic streams: A framework for composable semantic interpretation of sensor data," in *Proc. EWSN*, 2006, pp. 5−20.

[16] S. C. Geyik, B. K. Szymanski, P. Zerfos, and D. Verma, "Dynamic composition of services in sensor networks," in *Proc. IEEE Int. Conf. Service Computing*, 2010, pp. 242−249.

[17] P. Bartalos and M. Bieliková, "Automatic dynamic web service composition: A survey and problem formalization," *Comput. and Inf.*, vol. 30, pp. 793−827, 2011.

[18] J. Bronsted, K. M. Hansen, and M. Ingstrup, "A survey of service composition mechanisms in ubiquitous computing," in *Proc. UbiComp Workshop*, 2007, pp. 87−92.

[19] T. Andrews et al. (2003, May). Business process execution language for web services (BPEL). [Online]. Available: http://www.ibm.com/developerworks/library/specification/ws-bpel/.

[20] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. (2001, Mar.). Web services description language (WSDL). [Online]. Available: http://www.w3.org/TR/wsdl.

[21] D. Martin et al. (2003, Nov.). OWL-S: Semantic markup for web services. [Online]. Available: http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/.

[22] D. Box et al. (2007, Apr.). Simple object access protocol (SOAP). [Online]. Available: http://www.w3.org/TR/soap/.

[23] J. Ibbotson, S. Chapman, and B.K. Szymanski, "The case for an agile SOA," presented at the Annu. Conf. ITA, Adelphi, MD, 2007.

[24] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations*. R. E. Miller and J. W. Thatcher, Eds. New York, NY: Plenum Press, 1972, pp. 85−103.

[25] U. Feige, "A threshold of ln n for approximating set cover," *J. ACM*, vol. 45, pp. 634−652, Jul. 1998.

[26] M. P. Papazoglou and W. van den Heuvel, "Service oriented architectures: Approaches, technologies and research issues," *VLDB J.*, vol. 16, pp. 389−415, Jul. 2007.

[27] R. Sugihara and R. K. Gupta, "Programming models for sensor networks: A survey," *ACM Trans. Sensor Netw.*, vol. 4, pp. 1−29, Mar. 2008.

[28] L. Mottola and G. P. Picco, "Programming wireless sensor networks: Fundamental concepts and state of the art," *ACM Comput. Surveys*, vol. 43, pp. 1−51, Apr. 2011.

[29] J. Cardoso, A. P. Sheth, J. A. Miller, J. Arnold, and K. Kochut, "Quality of service for workflows and web service processes," *J. Web Semantics*, vol. 1, pp. 281−308, Apr. 2004.

[30] S. Kona, A. Bansal, and G. Gupta, "Automatic composition of semantic web services," in *Proc. ICWS*, 2007, pp. 150−158.

[31] A. V. Riabov, E. Bouillet, M. D. Feblowitz, Z. Liu, and A. Ranganathan, "Wishful search: Interactive composition of data mashups," in *Proc. ACM Int. Conf. World Wide Web*, 2008, pp. 775−784.

[32] E. Sirin, B. Parsia, D. Wu, J. Hendler, and D. Nau, "HTN planning for web service composition using SHOP2," *Web Semantics: Sci., Services and Agents the World Wide Web*, vol. 1, pp. 377−396, Oct. 2004.

[33] E. Sirin, B. Parsia, and J. Hendler, "Filtering and selecting semantic web services with interactive composition techniques," *IEEE Intell. Syst.*, vol. 19, pp. 42−49, Jul. 2004.

[34] E. Sirin, J. Hendler, and B. Parsia, "Semi-automatic composition of web services using semantic descriptions," in *Proc. WSMAI*, 2003, pp. 17−24.

[35] R. Zhang, I. B. Arpinar, and B. Aleman-Meza, "Automatic composition of semantic web services," in *Proc. ICWS*, 2003, pp. 38−41.

[36] F. Lécué, A. Delteil, A. Léger, and O. Boissier, "Web service composition as a composition of valid and robust semantic links," *Int. J. Cooperative Inf. Syst.*, vol. 18, no. 1, pp. 1−62, 2009.

[37] D. van Thanh and I. Jorstad, "A service-oriented architecture framework for mobile services," in *Proc. IEEE Telecommunications*, 2005, pp. 65−70.

[38] W. Tan, Y. Fan, M. Zhou, and Z. Tian, "Data-driven service composition in enterprise SOA solutions: A Petri Net approach," *IEEE Trans. Autom. Sci. Eng.*, vol. 7, pp. 686−694, Jul. 2010.

[39] X. Wang, J. Wang, Z. Zheng, Y. Xu, and M. Yang, "Service composition in service-oriented wireless sensor networks with persistent queries," in *Proc. CCNC*, 2009, pp. 1−5.

[40] J. W. Branch et al., "Towards middleware components for distributed actuator coordination," presented at EmNets, Boston, MA, 2006.

[41] A. Bamis, N. Singh, and A. Savvides, "An architecture for dynamic reconfiguration of data flows in sensor networks," Yale Univ., New Haven, CT, Tech. Rep., 2007.

[42] L. Luo, T. F. Abdelzaher, T. He, and J. A. Stankovic, "EnviroSuite: An environmentally immersive programming framework for sensor networks," *ACM Trans. Embedded Comput. Syst.*, vol. 5, pp. 543−576, Aug. 2006.

[43] A. Bakshi, V. K. Prasanna, J. Reich, and D. Larner, "The abstract task graph: A methodology for architecture-independent programming of networked sensor systems," in *Proc. EESR*, 2005, pp. 19−24.

[44] W. Heinzelman, A. Murphy, H. Carvalho, and M. Perillo, "Middleware to support sensor network applications," *IEEE Network*, vol. 18, pp. 6−14, Jan. 2004.

Sahin Cem Geyik is currently an engineer and applied scientist at Turn Inc., Redwood City, CA. He received his Ph.D degree in Computer Science from Rensselaer Polytechnic Institute (RPI), Troy, NY, in 2012, and his BS degree in Computer Engineering from Bogazici University in Istanbul, Turkey, in 2007.

Boleslaw K. Szymanski (M'82-F'99) is the Claire and Roland Schmitt Distinguished Professor of Computer Science and the Founding Director of the Center for Pervasive Computing and Networking at Rensselaer Polytechnic Institute (RPI), Troy, NY. He received his Ph.D. degree in Computer Science from the National Academy of Sciences, Warsaw, Poland, in 1976.

Petros Zerfos Petros Zerfos is currently a research staff member at IBM T.J. Watson Research Center. He received his Ph.D. and M.Sc. in Computer Science from UCLA in 2005 and 2003 respectively, and a B.Eng. in Electrical Computer Engineering from the National Technical University of Athens (NTUA), Greece in 1999.