Proc. 7th IEEE International Conference on Services Computing (SCC 2010), pp. 242-249, Miami, FL, July 5-10, 2010.

Dynamic Composition of Services in Sensor Networks

Sahin Cem Geyik and Boleslaw K. Szymanski
Department of Computer Science
Rensselaer Polytechnic Institute
Troy, NY, 12180
Email: {geyiks,szymansk}@cs.rpi.edu

Petros Zerfos and Dinesh Verma
Wireless Networking Lab
IBM T.J. Watson Research Center
Hawthorne, NY, 10532
Email: {pzerfos, dverma}@us.ibm.com

***Abstract*—Service modeling and composition is a fundamental method for offering advanced functionality by combining a set of primitive services provided by the system. Unlike in the case of web services for which there is an abundance of reliable resources, in sensor networks, the resources are constrained and communication among nodes is error-prone and unreliable. Such a dynamic environment requires a continuous adaptation of the composition of services. In this paper, we first propose a graph-based model of 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, which is NP-complete. We then propose two heuristic methods for its solution, the top-down and the bottom-up, and discuss their centralized and distributed implementations. Using simulations, we evaluate their performance.**

***Keywords*- Service Composition, Sensor Networks, Service Modeling.**

I. INTRODUCTION

A wireless sensor network is an ensemble of low-cost devices that collect raw measurement data from the environment, transform it through a series of operations into more meaningful aggregate values, and relay these values (possibly over multiple hops) to base stations, for collection and further processing by end-users [1]. Due to limited communication bandwidth, node processing and energy resources, sensor network applications are implemented and run distributedly 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 to the end-user. For example, the tracking and object identification application of Figure 1 is provided by combining acoustic localization, object identification algorithms and camera-based visual tracking functionalities.

Early programming frameworks for sensor applications [2][3] recognized the need for a component-based design that compartmentalizes at the source-code level the transformational steps that measurement data must undertake. Recent advances in this area further propose the use of a high-level language such as Haskell [18], WaveScript [4] or Prolog [13] to describe the interconnection of application components, each of which is implemented in a lower-level, device-specific language. However, prior efforts provide a static description of the sensor network application that does not allow for a flexible re-engineering of the application at *runtime* that makes the service resilient to failures and disconnections. Furthermore, they are ill-suited for the dynamic sensor network environment, and do not adapt the service model to the ever-changing processing and energy resources of the nodes. The goal of this work is to propose a modeling and composition framework for sensor services that allows for adaptation of the service descriptions to the dynamics of the underlying sensor network deployment.

In this paper, we take a service-oriented approach for representing sensor network applications and view them 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 of the service deployment that includes processing and communication costs.

Based on this modeling approach, we formulate the process of dynamic sensor service composition as a cost-optimization problem, which can be shown to be NP-complete. Two heuristic approaches, the top-down and the bottom-up, are then proposed to solve this problem, which differ on how the composition process proceeds: from the composite service description to the identification of the primitive components that are required to be provisioned, or vice versa. Centralized and distributed implementations are also discussed.

In summary, this paper makes the following contributions:

- a modeling framework for sensor services that follows a natural data flow graph formulation enhanced with metadata information, which is amenable to analysis;
- a formulation of the sensor service composition process as a cost-optimization problem;
- two algorithms that use heuristics for performing automatic service composition that adapts to the dynamics of the sensor network environment;
- simulation results that provide a preliminary analysis of the performance of these algorithms.

The rest of the paper is organized as follows. Section II

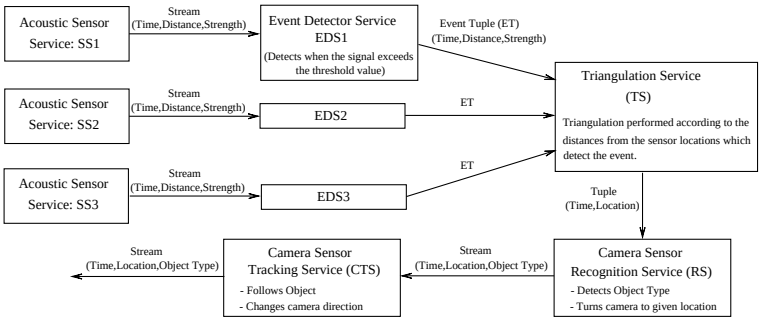

Figure 1. A Composite Service Example

outlines a model of sensor services and formulates the problem of sensor service composition. Section III describes the two approaches that we propose for solving that problem and discusses their centralized and distributed implementations. Performance is analyzed through simulations in Section IV and in Section V related work is presented. The paper concludes in Section VI with our ongoing work.

II. Sensor Service Modeling and Composition

We start with introducing a graph-based model for representing sensor network services that is intuitive and also amenable to analysis. 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 provides additional information that characterizes the service and its outputs:

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

Although the inputs and outputs of a sensor service change with time, to abbreviate the notation, we omit the t subscript, which instead is implied. A service implemented in a sensor network may be just a $source\ service$, which does not receive any input and only outputs data. Similarly, we can also have $sink\ services$, which do not produce any outputs (but may take an action instead).

In the above service definition, $metadata$ carries information about the data and the services that process it, following the approach in [5]. Metadata may contain properties of the data such as levels of reliability, as well as cost information and certain characteristics of the service itself, such as energy consumption per output data produced, processing delays, number of other services that make use of its outputs, etc. The service metadata depends on time (t) and each service has different types of metadata that is transmitted to other services offered by the sensor network. As it will be shown in the later sections, metadata information is used to find which services are most cost-efficient to use for a given composite service requested by an end user.

A. Service Graph of a Sensor Network

Service graph of a sensor network, G_S, is a graph in which vertices represent services and directional edges represent the potential data flows between services. The edge directed from the vertex of service A to the vertex of service B is created if and only if the output of A and input of B intersect in some fields. That is, informally, A can provide some of its products (output) to the use of B through this directional edge. Additionally, we require that each input of B is connected to at least one output of some service (we assume that by definition, a service requires all its input data to produce any output). A formal definition of the service graph is given below:

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

$V = \{s_i\}$ (one vertex per service) and

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

Although not stated explicitly in the graph definition, we require matching properties of the metadata of the service providing some data as output with those of the service that is requesting this data as input. For example, if a service requires input with reliability of at least a certain value, 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 requires exact match between input and output fields of two services to create an edge between them, this requirement can be relaxed by using type hierarchies. If a service A's output field is a subtype of a service B's input field, then A can provide the information that B requires, hence the edge between them should be allowed. An example is a type *vehicle* with a subtype *car*. If a service can identify an object as a car, then by the subtype-type relationship, it has automatically classified it as a vehicle.

B. Definition of the Service Composition Problem

In our model, there are two basic types of costs related to service composition: first, the processing cost of each service, for example the energy cost incurred by activating an implementation of a service. Second, the cost of communication between two services needed to exchange information (e.g., transfer delay between two services or load on the network due to service communication). This cost is interpreted as the edge costs in the service graph.

Service composition requires finding such a subset of services $S_c \subset S$ and data flows between them that each service in S_c has its inputs provided by at least one service in S_c that is capable of sending data to it. Furthermore, 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 and satisfying certain properties:

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

Service composition may be considered as the task of finding a certain subgraph of the service graph (G_S) wherein only a subset of the possible edges (data flows) and vertices (services) is used. Furthermore, 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,

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

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

According to this definition, an edge $e_{i,j}$ cannot be a part of the composition scheme unless both V_i (representing service i) and V_j (representing service j) are selected for the composition. This problem formulation makes the composition scheme subject to changes in time since an optimal composition is dependent on the network conditions at time t. Services learn the network conditions via the metadata mechanism.

Although not shown here due to lack of space, we can also prove that even a very simple version of the above problem is NP-complete, because any instance of the set cover problem can be transformed to an instance of the Service Composition Problem in polynomial time. In the next section, we present heuristic algorithms for its solution.

III. Heuristics for Dynamic 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. We introduce two approaches termed *top-down* and *bottom-up*, with the latter one further divided into two variants, which differ in the amount of information used to find the composition. To ensure that the heuristics terminate, we consider only service graphs that are *acyclic* and *directed*. We leave the consideration of service composition costs on more general graphs to future work.

A. 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. Then, the services that were selected choose their input providers and so on. A key requirement in this scheme is that all services at the current level are composed before any services on the next level are considered. 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 involved in the composition process.

At each level, first, we select the $critical\ services$, which are those that exclusively provide input fields of a service that are not provided by any other service. Since these services have to be included in any feasible set of services, they may cover some inputs that could also be provided by otherwise non-critical services. Then, a set of services is chosen such that among all sets that can supply the inputs to the service under consideration, the chosen set exhibits the smallest cost. To make such choice, we use a well-known heuristic for the set cover problem, which chooses the service that adds the smallest cost per each input covered. 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, as each service knows only its immediate neighbors.

B. Bottom-up Approach

The bottom-up approach sorts topologically the directed and acyclic service graph G_S. At each service, assuming the possible input providers (neighbors) have composed themselves, a subset of neighbors are chosen to satisfy input fields. A filtering function can be applied at this stage to filter out neighbors with unwanted properties (e.g. high cost, low reliability etc.).

The bottom-up method (Algorithm 1) is designed following the heuristic for the set cover. At each step, it attempts to find the neighboring service that has the smallest cost per new input field that it can provide. The cost is calculated by the composition graph of the neighbor node and communication cost between the neighbor and the

Algorithm 1 Algorithm for Choosing Services to Cover Input Set

method *find_comp(input,output_sets,compositions,S)*
$remaining\_in = input$
$remaining\_out = output\_sets$
$composition\_graph = vertex(service\ S)$
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 $\frac{gr\_cost(compositions[S_{min}])+comm\_cost(S_{min}->S)}{|remaining\_in \cap S_{min}|}$ is smallest
    end for
  end if
  $remaining\_in- = remaining\_in \cap S_{min}$
  $composition\_graph+ = compositions[S_{min}]+ edge(S_{min} \rightarrow S)$
  $remaining\_out- = S_{min}$
  for each service S_k in $remaining\_out$ **do**
    $compositions[S_k]- = compositions[S_k] \cap compositions[S_{min}]$
  end for
  if $remaining\_out == \emptyset$ **then**
    break
  end if
end while
return $composition\_graph$

service that is being composed. Note that, when a service is selected, its composition graph ($compositions[S_{min}]$) is subtracted from the graphs of all services that haven't been used yet. This is done in order to prevent duplicate additions of services and links between services. When a service A is chosen to provide input to another service B, all services and their links that denote information flow are already used in the composition. At a later step, when another service C is chosen to provide input to B, the common services and links between A and C are not counted towards the cost of C. As in the top-down approach, critical services are always included first due to the input fields that exclusively provide.

The bottom-up method, as described above, incurs significant communication overhead when implemented in a distributed manner, due to transferring complete composition subgraphs among the neighbors of a service. An alternative approach transmits upstream only the composition cost of the subgraph. When a service S chooses a set of services to utilize in order to satisfy its input, the cost of this service is sent upstream to services that may utilize S's output. Sending the collective cost information incurs a lighter load on the system. However, less information is also made available about the composition of a service's possible input providers and hence may result in a less cost-efficient composition. When a service does not know which services will be utilized by its input providers, it is not possible for it to reuse the services, potentially increasing the composition's cost.

C. Implementing the Composition Selection Algorithm

The selection of composition can be made either in a *centralized way*, at a single node that receives information from all services, or *distributively*, wherein each node with a service assign to it chooses which services it will use to satisfy inputs of its service. In the following, the details of these two approaches are further discussed.

1) Centralized Implementation: uses a single node on which metadata of each needed service is first collected. Then, the node runs either the top-down or bottom-up algorithm and selects the component services to be activated. Although not shown here due to lack of space, there are cases in which the top-down approach gives a lower cost solution than the bottom-up one, while in some others cases, the opposite is true. The service graph can be generated at the central node based on the sensor network topology, which is discovered through the information collected from every sensor node. This information includes the set of services (and their metadata) that the reporting node offers and the communication costs from that node to its neighborings.

2) Distributed Implementation: makes each node with a service allocated to it to select, independently of others, services that it needs to receive inputs for its service. The advantage of this scheme is its robustness to network faults and the quick reaction to changes in the network conditions. Additionally, no single node is selecting the entire composition, avoiding a single point of failure and bottleneck. The composition process proceeds from bottom to top. In the distributed implementation, a node selecting services has only information about its own neighbors, i.e. the services which could provide input to its service. This information is included in the metadata that the node receives from its potential input providers, as described in Section II.

When a composition process starts, messages are sent from the node at which the end-user request for service originated (considered to be at the top level of the service graph) to the nodes capable to provide 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 nodes with services whose outputs can satisfy the request will be considered neighbors of the node from which the user-request originates. This process repeats until the necessary information is disseminated downstream to all the nodes with source services, which do not have any incoming edges, i.e. any inputs. At that time, the reverse dissemination process takes place, in which, at each stage, the service composition algorithm is executed. Once every node with the

service is aware of its smallest composition cost, backward messaging takes place, where services on certain nodes are activated. This generates the composition graph.

Mapping the sensor network topology to the service graph is complicated in the distributed case. Finding which services can send their data to a given service, as well as the information on the cost of these services, requires every node to have global knowledge of the entire topology. Hence, a mechanism for distribution of service metadata among nodes is required. Once each node with a service knows its neighboring services in the services graph[1], the service costs can be exchanged between the nodes with services. The distribution of service metadata enables the distributed implementation of the bottom-up approach. To disseminate service costs among nodes in the sensor network, a simple protocol like controlled flooding can be used. Furthermore, such a scheme can also be easily modified to eliminate costly links between services. For example, if the node on which a service A is implemented is more than α hops away from a node that provides another service B, then these two services might not be regarded as neighbor services. By using *time-to-live* when transferring service costs, a virtual service graph in the sensor network with a time or hop limit can be created.

D. Dynamic Composition

In a sensor network environment, the conditions of service can change fairly frequently. Hence, it is important to be able to dynamically change service composition. Metadata information exchanged throughout the network can be used for this purpose. For the centralized implementation, dynamic composition is similar to generating the initial one: for each update in the system, the composition process is re-executed, triggered by the new costs. In the distributed case, each node with a service will decide on its new composition graph based on the updates of its neighbors. When a node with a service updates its own information, it also notifies its neighbors so that they can, if needed, change their respective compositions. If a service (or a link of a service) is not used anymore, the node running will deactivate its corresponding output links. If all links of the service are deactivated, the node will stop the service itself and will send stop signals to every service that provided input to stopped service.

Finally, an important issue that needs to be considered in dynamic composition is the frequency of updates. Some of the metrics, such as residual energy on the nodes that the services are implemented on, change continuously. Therefore a dynamic composition solution would be triggered constantly by the change, leading to high overhead. Such a situation can be prevented by setting up a composition update period, or an updating scheme. A node running a service can wait for significant changes in service conditions to notify the central node or the nodes with neighboring services. Example of such significant changes are threshold on reliability, critical low battery level of the sensor node, etc.

[1]It should be noted that neighor relationship of services in the services graph does not necessarily imply that the nodes that are implemented on are neighbors in the network topology.

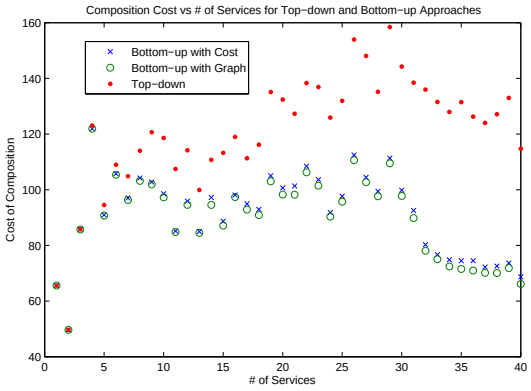

Figure 2. Comparison of Composition Costs incurred by the Top-down and Bottom-up Approaches

IV. Performance Evaluation

We conduct simulations to assess the performance of the proposed algorithms, both when used for deriving the initial composition of a service (Section IV-A), as well as for dynamic composition (Section IV-B).

A. Evaluation of Initial Composition

We evaluate the costs of composition incurred by each algorithm, given an initial set of services and their costs, as well as user-requests. For simplicity, activation and communication costs are combined into a single metric that could correspond for example to the total energy spent by the algorithm.

To evaluate the composition performance of our algorithms, we created 10,000 distinct cases of services. Of these cases, 40 are basic ones, each with a unique number of services varying from 1 to 40. Moreover, 250 variants of each basic case are generated. In each variant, we assigned each service a uniformly distributed cost between 0 and 40, while the communication cost between services is also uniformly distributed between 0 and 50. First, the acyclic graph is created and then inputs and outputs are assigned according to the edges that have been created. A random user-request is chosen from the set of inputs and outputs defined in the graph. Then, our algorithms are run on this setting and performance results are collected.

Figure 2 shows that the bottom-up approach creates compositions with lower cost than the top-down approach. Furthermore in the bottom-up algorithm, sending the graph information upstream instead of collective cost information leads, as expected, to a lower cost of the final compositions.

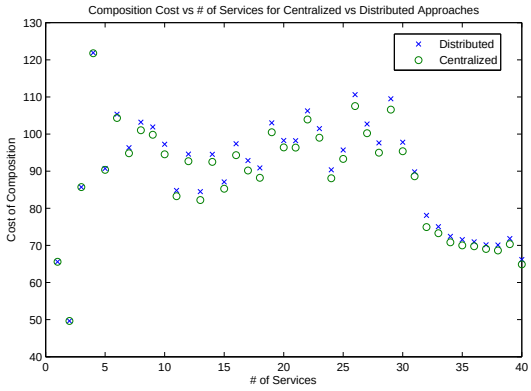

Figure 3. Cost of Composition Comparison of Centralized and Distributed Approaches

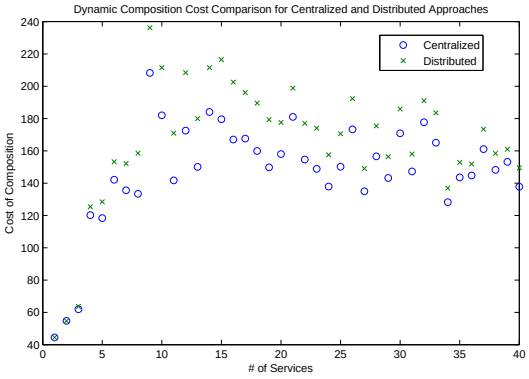

Figure 4. Comparison of Composition Costs for Dynamic Composition by Centralized and Distributed Approaches

In Figure 3, the costs of composition for the centralized and distributed approaches are compared. The centralized approach implements both the top-down as well as the bottom-up algorithms, and thus produces better results compared to the distributed one, which implements only the latter one (bottom-up). It is interesting to note that, although the top-down approach is much worse on average (as seen in Figure 2), it still has a certain advantage over the distributed one. This is due to the lower costs that the top-down algorithm incurs compared to the bottom-up one in certain cases, as mentioned before.

B. Evaluation of the Dynamic Composition

In this section we are presenting the dynamic composition results of centralized and distributed approaches. As mentioned before, the distributed approach selects composition based on the local information. Thus, a change in the local system conditions, such as service or communication cost change, deactivation or reactivation of services, immediately triggers the re-composition of a requested service. In the centralized approach however, there is a period of re-composition during which all the services update their information in the central decision making node and the actual re-composition is performed once this time period is over.

We have used a subset of the services generated for the experiments described in the previous section. As previously, service costs are uniformly distributed between 0–40, and communication costs between services are uniformly distributed between 0–50. We introduce events to this setting, which change these costs and activation states of the services. For example, every 0–20 (uniformly distributed) seconds a random service or a random link is assigned a new cost, from 0–40 and 0–50 ranges, respectively. We also change which subsets of the initial services are still active by choosing a random service in 0–50 second time periods and deactivating it for a period of 0–200 seconds. The centralized approach is set to perform a re-composition every 10 seconds and dynamic composition triggers a re-composition locally if a service's or link's cost changes significantly. Reactivation or deactivation of services also trigger a recomposition process. We set the simulation time of 3000 seconds for each experiment and service set.

In Figure 4, we present the cost of the composition of both the centralized and the distributed approach, averaged over the service operation. It can be seen that the centralized approach gives lower costs for each case for two reasons. First, the centralized approach has more information and generally creates compositions with lower cost than the distributed approach does. The second reason is the triggering mechanism of distributed approach; to lower the information overhead, a recomposition is performed only when there is a significant change in the service costs, and this results in the higher composition cost than necessary.

Figure 5 shows the activation ratio of services for both approaches, defined as the percentage of the simulation time that the user-request service was active. The user-request is considered active when all services in the composition graph are satisfied in terms of the inputs that they require. We observe that, for small number of services, the centralized approach gives a higher activation ratio, which means that the reactive re-composition of the distributed approach is not always helpful, as it is less likely to find the alternative routes locally. However, the centralized approach recomposes the services from scratch, hence creating an active composition more readily. From the figure, it can also be seen that the distributed approach gives better results for a large number of services. This means that the local reactive approach of distributed composition works well in finding alternate information flows dynamically.

To summarize, the distributed approach exhibits higher activation ratio, provided that the alternative compositions exist for the reactive means to work well. A trade-off exists since we have lower cost service composition schemes

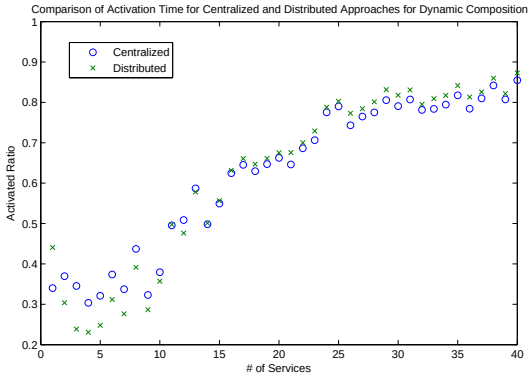

Figure 5. Comparison of Service Activation Ratios for Dynamic Composition by Centralized and Distributed Approaches

generated in the centralized approach.

V. Related Work

Service composition has enjoyed a continuous attention in the web services domain, where there are no constraints on resources and there is high user interactivity. The latter helps with semi-automated and manual composition approaches weakening the need for automated composition techniques. There are several detailed surveys on web services and composition [6] [7], and below we discuss in more detail efforts that are more closely related to our work.

The authors of [8] and [9] use OWL-S (Web Ontology Language - Services) language to describe the web services with their inputs and outputs. However their methods are user supervised in the sense that, at each step, a set of possible matchings for the user functionality needs are presented to the user who selects one or more services for use.

In [10], MARIO (Mashup Automation with Runtime Orchestration and Invocation) is introduced, which 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. In [11], the authors propose the use of service equivalence (both semantical and syntactical equivalence) in order to replace services in a mobile network where the connections between nodes are changing rapidly.

In Mao et al. [12], a dynamic web service composition method is proposed that considers quality of service (QoS) and network characteristics. Their method, $Automatic\ Path\ Creation$ service (APC), is centralized and looks for a shortest path from the end-user to the primitive services. This is similar to what we would like to achieve. However, our method is amenable to distributed execution. Furthermore, [12] does not consider the case where a subset of the output of a service can be used as input to another service.

In sensor networks, few approaches have been proposed for service composition. A significant one is [13] in which the authors provide a method based on logical programming through backward chaining for chaining 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 paper in the sensor networks domain [14] tries 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 [15], 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. Their system still requires user interaction.

A model similar to our service graph can be found in [16], 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 construction or cost measures. Another work worth mentioning is that of [17]. The authors present MiLAN, which is a middleware for sensor applications. MiLAN receives the application requirements in terms of the information needed 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.

VI. Conclusions and Ongoing Work

In this paper a novel method of service modeling and dynamic composition is described, which is suitable to the unreliable and dynamic sensor network environments. Two heuristic algorithms for composition of sensor services are introduced that differ in the direction of traversing the service graph during the composition process. Centralized and distributed implementations of these algorithms are also described and evaluated through simulations, along with the associated trade-off between overhead of performing the composition and the cost of the final composition result.

The implementation and evaluation of the algorithms proposed in this paper in real systems is the main focus of our ongoing work. Two runtime environments are targeted for this purpose: the stream processing language called SPADE [19] of *System S*, the large-scale stream processing

system developed by IBM Research, and the ITA Sensor Fabric [20]. In SPADE, each processing unit is modeled as a stream, with predefined inputs and outputs, similar to our definition of a service. Streams are linked with one another, at present only in static service graphs to provide composite services. We are planning to extend the SPADE framework to dynamically activate/de-activate streams according to our proposed composition algorithms, thus emulating dynamic composition. Metadata, which can be supported by SPADE as separate streams, can be used to trigger dynamic re-compositions.

Our second environment for implementation is the ITA-developed Sensor Fabric [20]. It provides a lightweight publish/subscribe framework that can interconnect sensor nodes over multiple hops. Message topics defined in the Sensor Fabric can be used to specify input and outputs of services that are offered by the nodes, as well as metadata. Connectivity among services on the nodes is determined from the topics that they subscribe to, which can be specified by our proposed composition algorithms.

ACKNOWLEDGMENT

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.

REFERENCES

[1] Akyildiz, I. F., Su, W., Sankarasubramaniam, Y., Cayirci, E., *A Survey on Sensor Networks*, Computer Networks, 2002, pp. 393-422.

[2] Gay, D., Levis, P., von Behren, R., Welsh, M., Brewer, E., Culler, D., *The nesC Language: A Holistic Approach to Networked Embedded Systems*, ACM SIGPLAN conference, 2003

[3] Greenstein, B., Kohler, E., Estrin, D., *A sensor network application construction kit (SNACK)*, in ACM SenSys Conference, 2003

[4] Newton, R., Toledo, S., Girod, L., Balakrishnan, H., Madden, S., *Wishbone: profile-based partitioning for sensornet applications*, in NSDI Conference, 2009

[5] Ibbotson, J., Chapman, S., Szymanski, B. K., "The Case for an Agile SOA", First Annual Conference of the International Alliance, Adelphi, MD, September, 2007.

[6] Papazoglou, M. P., van den Heuvel, W., *Service oriented architectures: approaches, technologies and research issues* , The VLDB Journal, vol. 16, no. 3, July 2007, pp. 389-415.

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

[8] Sirin, E., Parsia, B., Hendler, J., *Composition-driven Filtering and Selection of Semantic Web Services*, In AAAI Spring Symposium on Semantic Web Services, 2004.

[9] Sirin, E., Hendler, J., Parsia, B., *Semi-automatic Composition of Web Services using Semantic Descriptions*, In Web Services: Modeling, Architecture and Infrastructure workshop in ICEIS 2003, Angers, France, April 2003.

[10] Riabov, A. V., Bouillet, E., Feblowitz, M. D., Liu, Z., Ranganathan, A., *Wishful Search: Interactive Composition of Data Mashups*, in WWW Conference, Beijing, China, pp. 775-784, 2008.

[11] van Thanh, D., Jorstad, I., *A Service-Oriented Architecture Framework for Mobile Services*, Proceedings of IEEE Telecommunications 2005, pp. 65-70.

[12] Mao, Z. M., Katz, R. H., Brewer, E. A., *Fault-tolerant, Scalable, Wide-Area Internet Service Composition*, Technical Report UCB/CSD-01-1129, EECS Department, University of California, Berkeley, California, USA, 2001.

[13] Whitehouse, K., Zhao, F., Liu, J., *Semantic Streams: a Framework for Composable Semantic Interpretation of Sensor Data*, EWSN 2006, pp. 5-20.

[14] Wang, X., Wang, J., Zheng, Z., Xu, Y., Yang, M., *Service Composition in Service-Oriented Wireless Sensor Networks with Persistent Queries*, Consumer Communications and Networking Conference, CCNC 2009, Las Vegas, NV, pp. 1-5.

[15] Bamis, A., Singh, N., Savvides, A., *An Architecture for Dynamic Reconfiguration of Data Flows in Sensor Networks*, Technical Report, ENALAB, Yale University, 2007.

[16] Bakshi, A., Prasanna, V. K., Reich, J., Larner, D., *The Abstract Task Graph: A methodology for architecture-independent programming of networked sensor systems*, in Proc. Workshop on End-to-end, sense-and-respond systems, applications and services, 2005.

[17] Heinzelman, W., Murphy, A., Carvalho, H., Perillo, M., *Middleware to Support Sensor Network Applications*, IEEE Network Magazine Special Issue, Jan. 2004.

[18] Mainland, G., Morrisett, G., Welsh, M., *Flask: Staged Functional Programming for Sensor Networks*, Proc. of the 13th ACM SIGPLAN international conference on Functional programming, pp. 335-346, 2008.

[19] Gedik, B., Andrade, H., Wu, K., Yu, P. S., Doo, M., *SPADE: The System S Declarative Stream Processing Engine*, Proc. of the ACM SIGMOD International Conference on Management of Data, pp. 1123-1134, 2008.

[20] Wright, J., Gibson, C., Bergamaschi, F., Marcus, K., Pressley., R, Verma, G., Whipps, G., *A Dynamic Infrastructure for Interconnecting Disparate ISR/ISTAR Assets (The ITA Sensor Fabric)*, Proc. IEEE/ISIF Fusion 2009, July 2009.