Proc. 4th Annual Conference of International Technology Alliance, ACITA'10, London, U.K., September 15-16, 2010

Dynamic Composition of Services in Sensor Networks and Its Implementation under Sensor Fabric

Sahin Cem Geyik, Boleslaw K. Szymanski, Petros Zerfos, Dinesh Verma,
Joel Wright, Christopher Gibson, and Caleb Vincent

***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 heuristics for its solution, the top-down and the bottom-up, and discuss their centralized and distributed implementations. Using simulations, we evaluate their performance. Finally, we describe our implementation of the composition algorithms on the Fabric, a sensor network middleware infrastructure, produced by IBM UK as part of the ITA research programme.**

***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. 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 [1] recognized the need for a component-based design that compartmentalizes the transformational steps that measurement data must undertake at the source-code level. Recent advances in this area further propose the use of a high-level language such as Haskell [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 heuristics, 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

Sahin Cem Geyik, Boleslaw K. Szymanski and Caleb Vincent are with the Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY, 12180.
E-mail: {geyiks,szymansk,vincec}@cs.rpi.edu
Petros Zerfos and Dinesh Verma are with IBM T.J. Watson Research Center, Hawthorne, NY, 10532.
E-mail: {pzerfos, dverma}@us.ibm.com
Joel Wright and Christopher Gibson are with IBM UK, Hursley Park, Winchester, Hampshire, UK, SO21 2JN.
E-mail: {joel.wright, gibsoncr}@uk.ibm.com

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.

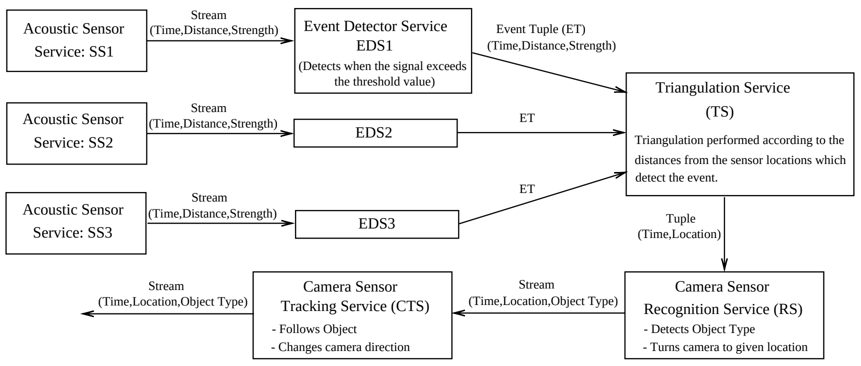
Acoustic Sensor Service: SS1
Stream (Time,Distance,Strength)
Event Detector Service EDS1
(Detects when the signal exceeds the threshold value)
Event Tuple (ET) (Time,Distance,Strength)
Acoustic Sensor Service: SS2
Stream (Time,Distance,Strength)
EDS2
ET
Triangulation Service (TS)
Triangulation performed 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

Figure 1. A Composite Service Example

versa. Centralized and distributed implementations are also discussed.

In summary, this paper makes the following contributions: (i) a sensor network service model that combines data flow graph representation with metadata information; (ii) the formulation of the sensor service composition process as a cost-optimization problem; (iii) two heuristics for performing dynamic service composition; (iv) simulation results for a preliminary analysis of these heuristics; and (v) description of the implementation of service composition in *Fabric* [15] middleware infrastructure.

II. 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, e.g., [3], and below we discuss in more detail efforts that are more closely related to our work.

The authors of [4] use OWL-S (Web Ontology Language - Services) language to describe the web services with their inputs and outputs. However their methods are user supervised as the user selects one or more services for use. MARIO (Mashup Automation with Runtime Orchestration and Invocation) [5] utilizes tags chosen by the user to provide possible composition schemes. 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 recomposition. In [6], 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. [7], 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, [7] 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 [8] 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 paper in the sensor networks domain [9] 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 [10], 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.

[11] 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 [12]. 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.

III. Sensor Service Modeling and Composition

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 [2]. 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. 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, consists of vertices representing services and directional edges representing 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 (informally, A can provide some of its outputs to B). 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

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

We also require that properties of the metadata of the service providing some data as output match with those of the service that is requesting this data as input.

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.

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[1] 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.

IV. Heuristics for Dynamic Service Composition in Sensor Networks

We introduce two approaches that are named *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*.

[1]We have proven that this problem is NP-complete

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. Clearly, this approach is a breadth-first traversal in the service graph, G_S, that 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. To select the remaining services needed for the composition optimally, we use a well-known heuristic for the set cover problem. It chooses the service that adds the smallest cost per each input field covered.

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 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 to avoid 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 are exclusively provided.

The bottom-up method, as described above, incurs significant communication overhead when implemented in a distributed manner, due to transferring complete composition

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 $composition\_graph$
```

subgraphs among the neighbors of a service. An alternative approach transmits only the composition cost of the subgraph upstream. 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 *distributedly*, wherein each node with a service assigned 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[2] and selects the component services to be activated. 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 neighboring nodes.

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 III.

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[3], 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.

[2]either of those can be optimal in certain cases

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

D. Dynamic Composition

To dynamically change service composition, metadata information exchanged throughout the network is used. 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.

V. Performance Evaluation and Fabric Implementation

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

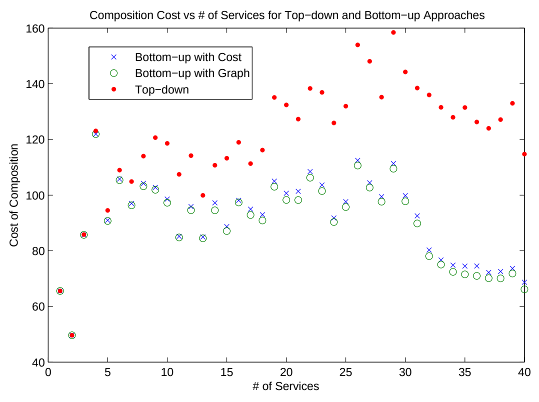

Figure 2. Comparison of Composition Costs incurred by the Top-down and Bottom-up Approaches

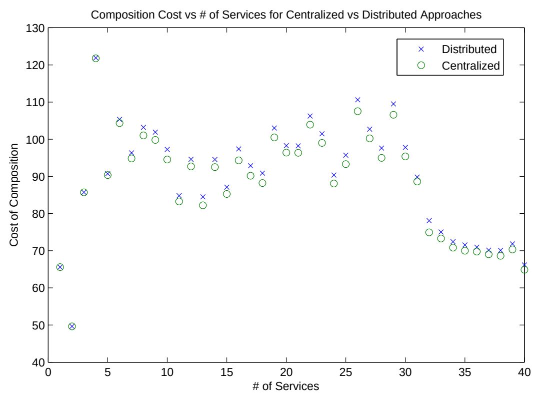

Figure 3. Cost of Composition Comparison of Centralized and Distributed Approaches

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.

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*

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 recomposition of a requested service. In the centralized approach however, there is a period of recomposition during which all the services update their information in the central decision making node and the actual recomposition 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. 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 recomposition every 10 seconds and dynamic composition triggers a recomposition 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 recomposition 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

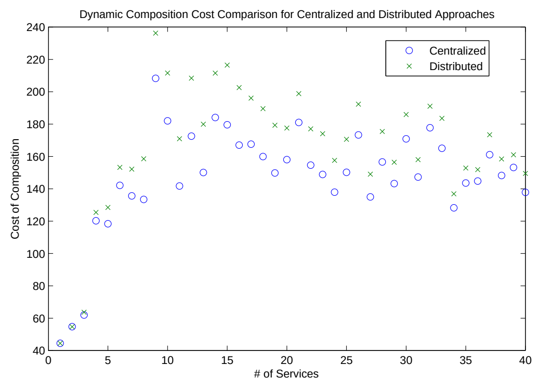

Figure 4. Comparison of Composition Costs for Dynamic Composition by Centralized and Distributed Approaches

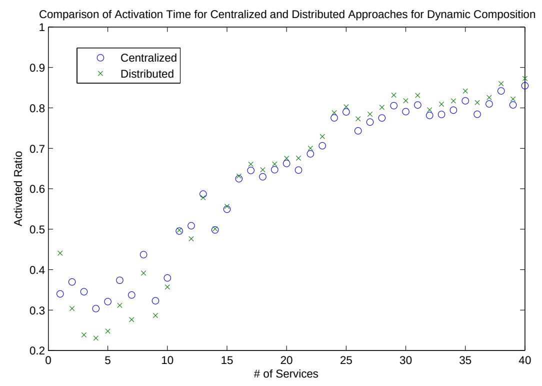

Figure 5. Comparison of Service Activation Ratios for Dynamic Composition by Centralized and Distributed Approaches

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 generated in the centralized approach.

C. Service Composition in the Sensor Fabric

The ITA Sensor Fabric, or Fabric, is a middleware architecture for sensor networks which is designed to simplify the development and operation of sensor network solutions, specifically those concerned with how sensors are attached, discovered, and utilized. In this section we describe our implementation of the service composition algorithms in this context, focusing on decisions regarding service modeling and the messaging structure for metadata transfer as well as detailing the key aspects of the design. For more information about the ITA Sensor Fabric, see [15].

For our implementation of service modeling and composition on the Fabric, we map our three types of service onto Fabric concepts as follows:

- **Source Services** - Because source services only output data, we represent them as simple Sensor Feeds,
- **Sink Services** - Sink services only consume data, and are therefore represented as actors subscribing to message feeds,
- **Intermediate Services** - Intermediate services both consume and produce data, and are represented as software services on the Fabric.

The Fabric allows for dynamic subscription to sensor feeds, and reaching of the service specific information (metadata) through querying the registry, hence it is easy for the service composition algorithm to perform the input provider selection. The publish/subscribe mechanism provided by the Fabric allows for the utilization of sensor feeds from either source services, or the intermediate services, which may perform data transformations.

We make use of the Fabric Registry to store and retrieve information regarding both the composition of services as well as the routing information, from which we determine the cost of information flow. The database stores information regarding services (e.g. security level, input/output, battery level etc.), and the overhead of metadata transfer for the service composition is actually the cost of distributing this information between the Fabric nodes.

Currently we implement our intermediate services as Fablets, software plug-ins running within the Fabric Manager which have the ability to both publish and subscribe to sensor feeds, and may apply any algorithm to incoming messages. Each Fablet receives and produces a set of preset inputs/outputs, defined in its Fabric Registry entry, and has the capability to query the Registry for input providers of the appropriate types. We employ a distributed request/response mechanism between services to determine their cost and metadata represented as semantic information.

We have chosen to use a hybrid of the top-down and bottom-up algorithms with the distributed design for our implementation in Fabric, since this was much more efficient in terms of activated services which assert traffic load onto the sensor network. At each step, the algorithm requests possible input providers for their composition information, and these providers recursively continue this process until no further inputs are required, i.e. the request reaches a source service. Hence a top-down request scheme transforms into a bottom-up notification of composition information at each step. This way, only the services which have the possibility of running compose themselves, as opposed to all services constantly trying to compose their input in the bottom-up approach. Hence we combine the efficiency of the top-down approach (needed for distributed implementation) with the composition performance (lower cost compositions) of the bottom-up approach. We also apply the set cover heuristic at each step for a service to choose a subset of its possible input providers to satisfy its input.

We use startup parameters to determine the specific details of a running service (such as privacy settings, inputs/outputs etc.), allowing us to have a single generic implementation of an intermediate service. Control messages handle service specific functionality, such as configuration messages, requesting metadata or performing compositions etc., while the information flow between services, such as sensor readings and processed data, are processed by the normal message handling features.

Our composition algorithm itself is implemented through a number of control message types, which determines the action to be taken based on the control message received. If this message is of type *compose_order*, the service must send its composition cost upstream to the requester through a *composition_info* message. Unless the service has already calculated its composition information, it gathers the required information, also by sending a *compose_order* message, to all of its potential input providers (discovered through the *find_possible_nodes* method).

When a service receives a *composition_info* message, it stores the cost information using the *add_new_info* method. Once cost information has been received from all possible input providers, the service calls the *select_calculate* method which uses a set cover heuristic to choose a subset of the inputs which fulfill the service's requirements. Finally, if the control message is of type *activate*, this means that this service has been chosen to be in the final composition, hence the service subscribes to the previously selected input feeds from the chosen subset of possible input providers.

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 [14] of *System S*, the large-scale stream processing system, as well as Sensor Fabric [15].

References

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

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

[3] 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.

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

[5] 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.

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

[7] 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.

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

[9] 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.

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

[11] 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.

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

[13] 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.

[14] 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.

[15] 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.

[16] *Really Small Message Broker*, http://www.alphaworks.ibm.com/tech/rsmb.

[17] Bent, G., Dantressangle, P., Vyvyan, D., Mowshowitz, A., Mitsou, V., *A Dynamic Distributed Federated Database*, Second Annual Conference of ITA, Imperial College, London, September, 2008.