Proc. 9th International Conference on Service Computing, SCC 2012, Honolulu, HI, June 24-29 2012

Policy-aware Service Composition in Sensor Networks

Raheleh Dilmaghani*, Sahin Geyik†, Keith Grueneberg*, Jorge Lobo*, S. Yousaf Shah†, Boleslaw K. Szymanski†, Petros Zerfos*

*IBM T.J. Watson Research Center, Hawthorne, NY, USA
†Rensselaer Polytechnic Institute, Department of Computer Science, Troy, NY, USA

***Abstract*—Sensor applications are typically composed of a number of functional components that run distributedly on the nodes of a sensor network, communicating and interacting with one another. Service composition is emerging as a viable approach towards the automatic synthesis of such sensor applications. However, for service composition to be practical, it has to comply with policies that define security and management constraints on the use of these service components and the interconnections amongst them. Prior research efforts have primarily focused on efficient evaluation of security policies during the composition process, which is not sufficient when generic network management constraints need to be expressed and evaluated. In this work, we propose a policy model and evaluation approach that enables us to define and check attribute-based policies, for controlling the sensor service composition process. Attribute-based policies are generic and allows us to express a wider spectrum of constraints than currently possible. Using this model and based on a previously-proposed sensor service composition algorithm, we introduce a policy evaluation method that allows for efficient checking of policy constraints. We further present a novel implementation of the proposed approach in the IBM Sensor Fabric, a middleware framework that simplifies the development of distributed, sensor network services. We also present preliminary performance evaluation results using our prototype.**

***Keywords*- sensor network services; service composition; attribute-based policies;**

I. Introduction

The structure of distributed sensor network applications and services [20] typically consists of a sequence of transformation and aggregation processing steps that execute on measurement data collected by the various sensing modalities of the sensor nodes. As such, it naturally lends itself to a service composition paradigm that is based on dataflow processing graphs [9], which can be further used to automate the synthesis of complex sensor services. However, such synthesis of a composite service through a combination of more primitive ones is usually subject to various constraints that are set by sensor network management and security systems, which are tasked with dynamically allocating network resources and regulating the sharing of the common sensor infrastructure among multiple applications that might run on it. Controlled sharing in particular becomes an important requirement in deployments where a common sensor infrastructure is contributed by multiple organizations with disparate administrative domains, each of which has its own policies with respect to how the sensor nodes and platforms will be used by the other partner organizations.

Service composition that is subject to given constraints has been extensively studied in the domain of web services. However, prior approaches are developed around the request-response interaction model of the web services environment and attempt to enforce either constraints imposed on individual component services (e.g. [2] [7]) or focus solely on security constraints and access control (e.g. [3] [10]). Recent work [14] looks at the evaluation of policies defined over information flows in a service composition, which seems to be more applicable to the dataflow model of sensor services, but the policy model assumed is limited and able to express only access control constraints based on multi-level security classes of information flows. While a step in the right direction, multi-level access control policies by themselves are not rich enough to express the management requirements of an operational sensor network infrastructure. For example, it is not possible to state conditions regarding the use of component services in a service composition graph based on metrics of resource utilization of the sensor nodes, a typically mandatory requirement in sensor network deployments that involve shared hardware, software and network components. The goal of the work described herein is to propose a policy model and service composition approach that overcome the aforementioned limitations.

In particular, our work is focused on addressing the following two challenges in sensor service composition: first, we are interested in expressing and evaluating *generic* policies that are able to capture the characteristics of the complete service graph, as opposed to the more narrow scope of information flows or individual service nodes/instances. Such policies might refer to aggregate characteristics of the composite service (e.g. "do not include in the service composition more than 3 instances of service *A* that are provided by organization *B*"), or span arbitrary service processing pipelines (also referred to as "service chains" in [14]), which might not necessarily be connected through an information flow (e.g. "do not include service instance *C* in the composition if service instances *A* and *B* are included", wherein services "A", "B" and "C" are not necessarily part of an information flow).

The second challenge that this work tackles is that of *efficiency* in policy evaluation: as a potentially large number of candidate composite service graphs might be generated as

a result of a composition request, the overhead of evaluating all the policy constraints for each service component selected, as well as on the overall graph, might be prohibitively high for the service composer. It is imperative that evaluation of policies be fast for practical service composition in a sensor network environment, in which dynamic re-composition of sensor services might take place often due to continuously varying node resources and connectivity.

To address these challenges, we propose a novel policy model and evaluation method that is centered around the concept of service composition graphs. We develop a resource model that exposes the characteristics of such composition graphs to a policy language [12], so as to enable authoring and evaluation of *attribute-based* policies. Attribute-based policies are able to express more general constraints than those based on multi-level security classes. The policy model is extensible and allows for incremental support of progressively more complicated constraints expressed over such graphs.

To make the evaluation of service composition policies efficient, we further propose a method for checking policy constraints during the service composition process, at runtime, as the service graph is being constructed. While the approach in [14] checks for compliance after all candidate composition graphs have been generated, our proposed method allows for *early elimination* of service instances that might violate policy constraints, which decreases the time needed for composition and policy checking and reduces the number of candidate service graphs that need to be checked. Both the policy model and the evaluation method, along with the sensor service composition algorithm, are implemented and evaluated in Sensor Fabric [19], a middleware framework for developing distributed sensor network applications and services. In summary, our work makes the following contributions:

- A policy model that is able to express *attribute-based* policies for sensor service composition based on dataflow graphs.
- A new policy evaluation algorithm that checks *at runtime* the service composition graph that is being constructed, leading to a faster service composition and policy check process.
- A novel implementation of our proposed police-aware service composition approach in Sensor Fabric, a software framework for sensor network application and services. Preliminary performance evaluation results are also presented.

The rest of this paper is structured as follows: Section II presents background information on the sensor service model that is assumed. Section III describes our proposed policy model within the context of the sensor service composition process. In Section IV, an novel method for evaluation of policies expressed in the proposed framework is outlined. Section V presents the design and implementation of the policy model and evaluation framework in a prototype system for developing sensor network applications. Using our prototype, we present preliminary performance evaluation results in Section VI. Related work is outlined in Section VIII, while Section IX concludes the paper.

II. Background

The work in [9] introduced a model for describing sensor applications that intuitively captures the aforementioned salient characteristics and interdependencies among the individual sensor services that comprise complete applications, and we adopt this model for the remainder of this work.

According to [9], a sensor service s_i is defined by the input data that it accepts denoted as an ordered list of typed fields, the transformation function that it applies to its input, the output data that it produces as an ordered list of typed fields, as well as metadata, which 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.

In the above service definition, $metadata$ carries information about the data and the services that process it, following the approach in [11]. Matadata 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.

A *composite sensor service*, i.e. a service that is provided through a suitable combination of a plurality of other, simpler services, is represented by a *service graph* G_S. The vertices of a service graph represent services and directional edges denote the potential data flows between them. 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 (i.e. the type of input field is same as the type of the output field). That is, informally, A can provide some of the data that it produces (output) for use by B through this directional edge. Additionally, we require that each input of B is connected to at least one output of some service. A formal definition of the service graph is given below:

$$G_S = \{V, E\} \text{ and } V = \{s_i\} \text{ (one vertex per service) and}$$

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

III. Policy Model

In this section, we present a model to define specifications of policy rules. This model is used in the context of service composition and handles constraints among directly and indirectly interacting services in the given service graph. In our proposed model we use the tuple representation of a service s_i with its input set, metadata and output set as

in Section II. As an example, metadata can include service node affiliation and type. Furthermore, policy rules define service selection criteria. Thus, we extend the previously proposed service composition model to check for policies at the time of composition as required by service descriptions [9]. We augment the composite sensor service model with the following definitions that will be used for policy evaluation:

Definition III.1. Service Arguments and Variables- *A service argument refers to a service input, output, or metadata field and is represented by the . notation after the service name. A* variable argument *is a service argument where the service name is replaced by a term of the form V[i], where i is a positive integer. We call V[i] a* service variable.

Example III.1. *$V[1]$ is a service variable that refers to any service. $V[1].ID$ is a variable argument that refers to any service with the field ID.*

Definition III.2. Valid argument- *A service argument is valid if that argument is defined for that service as a field.*

Definition III.3. Compatible Services- *Two services are compatible if at least one of the output fields of one service intersects with at least one of the input fields of the other.*

As a working example for the illustration of the policy model and evaluation method that is following, consider the composite service graph of Figure 1, which illustrates a hypothetical sensor application used for "object identification and tracking", as originally presented in [9]. Audio measurements from three acoustic Sensor Services, SS are collected by an Event Detector Service, EDS, and are used to localize the source of sound through a Triangulation Service, TS. The results of triangulation are then transmitted to a camera Recognition Service, RS, to identify the type of the object that was the source of sound. The output of the camera recognition service is finally fed to the Camera sensor Tracking Service, CTS, to allow for the camera to tilt towards the direction of the object.

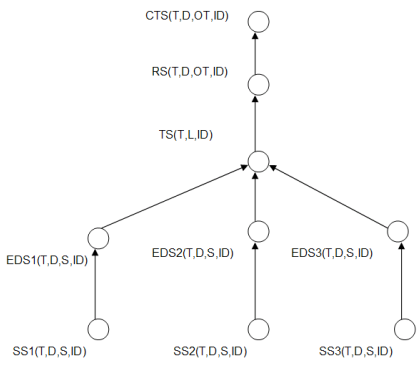

Fig. 1. Composite service graph of the "object identification and tracking" sensor service.

Using the above extended service model, we define policy rules as follows:

Definition III.4. *Policy Rule- A policy rule is a triplet comprising of $[Arguments, Constraint, Service\ Node]$ where:*

- *Arguments are a set of service or variable arguments as defined in III.1.*
- *Constraints are equality or inequality conditions involving arguments that refer to metadata fields or constants.*
- *Service node is a service or a service variable.*

Given a policy rule, an instance of that rule is the policy rule that results after replacing every service variable with the name of a service. An instance of a policy rule is valid if all the arguments in the instance are valid.

Example III.2. *Given a policy rule of $[V[1], V[1].ID = US, V[1]]$, an instance of the policy rule is $[SS1, SS1.ID = US, SS1]$. Another instance is $[CTS, CTS.ID = US, CTS]$, etc. This policy define a rule that only adds services that are US affiliated.*

Example III.3. *$V[1].D$ is a valid argument for services of type SS, EDS, RS, and CTS but it is not a valid argument for service type TS. Hence, given a policy rule of $[V[1].D, V[1].ID = US, V[1]]$, there is no valid instance of the rule for service type TS.*

Definition III.5. *Composition path- A composition path is a set of directed edges among compatible services. Given a set of n services, $S_1, S_2, S_i, S_{i+1}, ..., S_n$ and sequence of $n-1$ composition operation among them, a composition path is represented as a set of pairs of compatible service arguments, such as $< S_1.O_a, S_2.I_b >, ..., < S_i.O_c, S_{i+1}.I_d >, ..., < S_{n-1}.O_e, S_n.I_f >$, where a, b, c, d, e, f, m, n are all integers, $a, c, e <= m$ and $b, d, f <= n$, and the services participating in the composition operation all appear on a path in the service composition graph.*

Definition III.6. *Applicable Policy Instances- Given a service composition graph and a policy instance, the instance is applicable if:*
a. the service node appears as a vertex in the service composition graph and
b. all the services mentioned in the arguments appear as vertices in the service composition graph and
c. there are composition paths in the service composition graph between the service node and every service mentioned in the arguments.

If no applicable policy exists, the node is selected for service composition as if there were no policy to be evaluated.

Definition III.7. *Violation of a policy- A node in a service composition graph violates a policy if there is an instance of that policy that is applicable and the constraint is violated.*

Table I presents examples of policies written on the composite service graph of Figure 1 using our proposed policy model. Policy 1, defines a rule that service node $EDS1$, the event detector sensor, can access any arguments of the acoustic sensor service tuple, $SS1$, if $SS1$ belongs to US. Policy 2 indicates access of triangulation service, TS to distance and

TABLE I
POLICY RULES

Number	Arguments	constraint	*service node*
1	$\{SS1.ID, SS1.D, SS1.S, SS1.T\}$	$SS1.ID = US$	$EDS1$
2	$\{EDS1.D, EDS1.S, EDS1.T\}$	$EDS1.T = given$	TS
3	$\{RS.L, RS.T, RS.OT\}$	$RS.T = given$	CTS
4	$\{RS.L, RS.OT\}$	$RS.L = given$	CTS
5	$V[1]$	$V[1].type = SS \wedge V[1].ID = US \wedge V[2].type = EDS$	$V[2]$
6	$\{V[1], V[2]\}$	$V[1].type = SS \wedge V[2].type = EDS \wedge V[1].D = V[2].D$	TS
7	$\{V[1].ID, V[1].S, V[2].L\}$	$V[1].type = SS \wedge V[1].ID = US \wedge V[2].type = TS$	CTS
8	$\{V[1], V[2]\}$	$V[1].type = A, V[2].type = B, V[3].type! = C$	$V[3]$

signal fields of $EDS1$ at a given time. Policy 3 defines access to sensor RS by camera tracking sensor, CTS, at a given time. Policy 4 defines access to arguments of RS service from a given location. Policy 3 and 4 taken together force RS to be located in given location and can only be used during the given time. Policy 5 defines a rule for EDS node type to access arguments of acoustic sensor node type, SS, for US affiliated acoustic sensors. Policy 6 is a policy defining a rule for triangulation service to access event detector type sensor and acoustic sensor when sensors are co-located. This policy defines a constraint on more than one node. Policy 7 also refers to global properties of the composite service graph and dictates that camera tracking service can access acoustic sensor types and triangulation service types, if data is from US affiliated acoustic sensor. Lastly, policy 8 defines a rule for a service node, such that if it depends on services of type A and B, then it cannot involve a service of type C.

The above policy model allows us to define *attribute-based* policies that state conditions about service compatibility that are more general compared to simple multi-level security classes, thus controlling potential flow of information in a more generic manner. It also enables us to express policies that are *global*, in the sense that they include constraints over a collection of the nodes across the composition graph that might not be necessarily connected. Finally, as seeing in Policy 8 above, using this model, policies that refer to *separation of responsibilities* can be expressed.

IV. POLICY EVALUATION

To evaluate a service composition against the given policies, we make use of the following definition:

Definition IV.1. Feasible composition- *A composition is feasible (policy-compliant) iff there is no violation of policies with respect to a given service composition graph.*

Algorithm 1 performs policy evaluation during the composition process, for every candidate service S that is being considered for selection in the service composition graph that is currently being constructed. It does so by determining whether adding this candidate to the current service composition will not violate any policy. The algorithm takes as input a node (the candidate service instance), the policy table, as well as the service composition graph that has been constructed up to that point, assuming that the candidate service instance has been added to the graph. It checks the applicable policies for the node, i.e. entries in the policy table where the node is either a service node or appears in the arguments. In either case, it returns false if the policy is violated in the service composition graph or returns true if none of the policies are violated.

Algorithm 1 *PolicyEvaluation(S, PT, G)*

```
for each policy P in PT do
   where S is the service node in P do
   if P is violated in G then
      return False
   end if
end for
for each policy P in PT do
   where S can be an argument in P do
   if P is violated in G then
      return False
   end if
end for
return True
```

Algorithm 1 is used in conjuction with a service composition algorithm, more specifically when service selection is performed. In the system and performance evaluation experiments that we describe in the following sections, we employ essentially the centralized version of the service composition algorithm of [9].

V. SYSTEM DESIGN & IMPLEMENTATION

We develop a novel implementation of the centralized service composition algorithm of [9] along with the policy model and evaluation algorithm described in the previous sections using the IBM *Sensor Fabric* [19] environment (Figure 2). The Sensor Fabric is a middleware architecture designed to simplify the development and operation of sensor network applications and services by automating tasks such as sensor node discovery, connection and management. It follows a Service Oriented Architecture (SOA) approach to sensor network development and provides a two-way messaging bus along with a set of middleware services for transparent handling of connectivity and routing among sensor nodes. Service descriptions, the composite service model as described in Section II, as well as infrastructure information about the Fabric are

stored in the Fabric Registry (Figure 2), a distributed, federated database [1] whose instances are attached to each Fabric node.

The centralized service composition capability of [9] was implemented as a "Fablet" service in the Fabric. A Fablet is a software plug-in running within the Fabric middleware, implementing a container abstraction through which the Fabric can be extended with new, user-defined sensor services. Fablets are event-driven, and apply processing logic as they receive incoming messages. The service composition fablet obtains the component service descriptions from the Fabric registry and runs the policy-enabled composition algorithm once it receives an appropriate "trigger" message from the sensor node's management module. A Policy Enforcement Point (PEP) that is implemented in the service composition fablet enforces the results of policies that are evaluated by the Policy Decision Point (PDP) that runs on the sensor node, as per Algorithm 1. These policies are obtained by the PDP from the policy repository (Figure 2), which may be either local, remote or even distributed [1]. The policy repository allows for dynamic modifications of the set of policies during the operation of the system. Once the (local) service composition process is completed, the composite service graph is committed back to the Fabric registry, for the sensor network management module to "re-wire" the service connectivity by enabling the selected services instances and setting up routing appropriately.

A runtime model for writing policies applicable to the service composition process (and evaluated by the PDP in the sensor fabric node as in Figure 2) was developed using the Policy Management Library (PML) framework [12], a Java-based framework for analysis, negotiation, distribution and evaluation of policies using the standards-based CIM-SPL [8] policy language. The runtime model represents a dynamic instance of the composite service graph and provides Java instance classes that describe the entities of service graphs such as nodes (i.e. service instances) and edges (i.e. connections among service instances), following the service model described in Section II. In the runtime model, a number of standard graph-based functions can be implemented, that compute certain characteristics of the composite service graph such as connectivity, path length, diameter, etc. Policies in SPL can make use of such methods implemented in the runtime classes that represent the service graph entities. We also make use of policy-attributes that are supported by PML, to select the subset of policies to be evaluated that are applicable to a given composite service subgraph, avoiding evaluation of all policies that might be stored in our policy repository.

As nodes are composed, the runtime model is updated with the new service instances (nodes) and connections that are selected by the service composition algorithm. This allows the evaluation of *global* policies that are applicable to the whole graph, and not just the current service node that is being composed. It also provides flexibility of executing policy evaluation in an *incremental* fashion, during the composition process, as well as after the service graph is fully composed. This distinguishes our approach from other recent proposals for policy-aware service composition such as [14], which assume availability of the fully composed composite service graph before evaluating any policies. Furthermore, it allows for early elimination of service instances that are not compliant with pre-specified policies. Upon completion of service composition, policies that require the full graph to be evaluated can now be checked. In the following section, we evaluate through experiments the additional overhead introduced by the new policy checking.

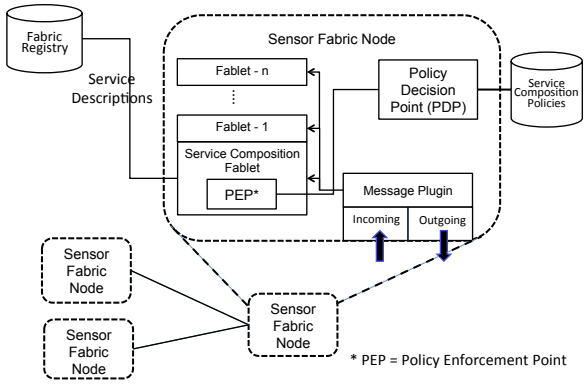

Fig. 2. Design of policy-enabled service composition in a sensor network middleware infrastructure (Sensor Fabric)

VI. Evaluation

Using the prototype system that we developed for policy-aware service composition in the Sensor Fabric, we conduct preliminary experiments to evaluate the performance of our approach. We seek to gain initial insights into the following two questions: (a) how much does policy checking contribute to a slow down in completing the service composition process compared to the baseline (i.e. without policies)? (b) how much does the composition cost of services increase as we limit the available options for service selection via constraints during the the composition process? Both questions are important in the context of sensor networks; on one hand, the dynamics of the sensor environment call for frequent re-compositions, which could be delayed due to the extra policy checking that is introduced. On the other hand, as sensor network resources are limited, we're interested in keeping the communication and processing costs of the composite service as low as possible, a task that depends on the availability of multiple services instances.

Experimental setup: we develop a utility application that is able to generate composite service graphs (i.e. descriptions of service instances and how they can be inter-connected with one another) of arbitrary size in the Sensor Fabric. The service graphs are laid out in a grid form that consists of N service levels with M services per level. The M services of the first (top) level in the grid represent the *sink* services, which are used as the user-requests to be automatically composed and which produce no outputs. The M service instances of the bottom level N are considered *source* services, which produce only outputs but have no inputs themselves. All other services

of the intermediate levels have a random number of inputs and outputs, each assigned a random data type. For all service levels, outputs of services of level i may provide data to the inputs of services of level $(i-1)$ if their respective data types match. The composite-service generator utility guarantees that only inputs and outputs of services of *adjacent* levels can be "connected" with one another, namely that they share data type. Additionally, the generator guarantees that there is always *at least one* service instance of level i whose output can provide data to an input a service of level $(i-1)$. Thus, a composite service graph for all user-requests (sinks) of the first service level always exists, with a "depth" of N.

For our experiments, we generate a 6×6 grid of service instances, with a maximum of 4 inputs and 3 outputs per service instance (randomly chosen), whose cost represents both processing and communication cost and is randomly chosen between 0 and 50. The types of the inputs and outputs of the services in the grid are randomly chosen from a set of 4 different types, so as to guarantee rich connectivity among the services of adjacent levels. The policies that we use for evaluation in our experiments are simple: each policy refers to an attribute of a *single* service instance in the grid, and can evaluate to true or false. Also, a single Sensor Fabric node stores all the service descriptions and their interconnections as generated by the composite service generator. This basic experimental scenario was followed deliberately: we seek to establish the *baseline* performance of runtime overhead that policy evaluation introduces to the service composition process, as well as to the cost of the composite service graph. Our on-going work includes more elaborate experiments, which include policies that refer to properties of the composite service graph, as well as topologies of multiple Sensor Fabric instances deployed across multiple machines.

Results: Figure 3 shows results of execution time (y-axis) required for service composition with policy evaluations, for different number of policies (x-axis) deployed in the service grid. Results are shown for different failure rates of the policies that have been deployed. Failed policies are those that are violoated and thus do not allow the selection of the node that they refer to. For each data point, results are averaged over three runs. As can be seen from the figure, there is an increase in the time that is needed to perform service composition as we increase the number of policies that need to be evaluated. More specifically, one can observe a linear trend, which is attributed to the fact that each policy that is evaluated refers to an attribute of a single service instance. In future work, we will evaluate the increase in execution time when we include policies that the evaluation depends on graph properties of the composite service (for example, the path-length among two service instances).

Figure 4 plots the total cost (y-axis) of the final composite service that has been generated by the service composition algorithm, as a function of the number of policies (x-axis) that the composition algorithm has to check. Results are again shown for various failure rates of policies, with each data point representing the average of three runs. As it is shown in the

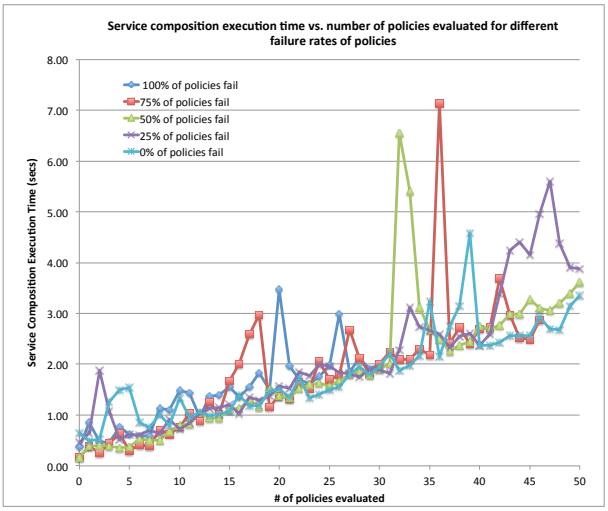

Fig. 3. Time to complete sensor service composition with policy checking for different number of policies evaluated, for different percentages of policy violations (failure).

figure, an increase in the number of policies that might fail leads to an increase in the cost of the composite service that the algorithm synthesizes. This is due to the fact that, as we introduce policies that fail (i.e. are violated), the service nodes that are referenced by them are excluded from being selected for composition. Consequently, the composition algorithm is forced to choose an alternative service instance that might not be as economical as one that could have been chosen if a policy that referred to it had not failed. This is also the reason for which, when there is a large number of policies that fail, composition of a given request might not even be feasible at all. This is also observed in Figure 4 as certain data points (particularly for high failure rates) are completely missing; the service composition run completely failed in that case and did not manage to generate a valid composite service graph.

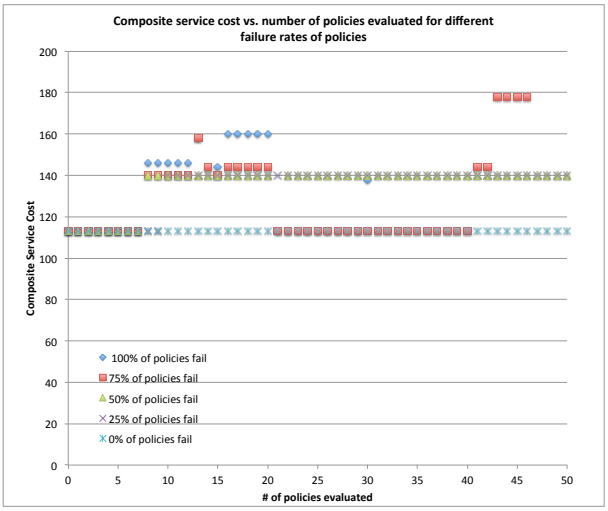

Fig. 4. Composite service cost produced with policy-aware service composition with different number of policies evaluated, for various percentages of policies evaluating to false (failure).

VII. Discussion

The early experience gained from designing and developing the policy-aware sensor service composition framework showed that our approach is viable and promising in evaluating attribute-based policies. However, few issues remain to be resolved and are the subject of our on-going work: first, while a more generic set of policies can now be expressed and evaluated over generic service graphs, there are still certain constraints that can be checked only *after* the composition process is over. For example, a constraint such as "include *at minimum* 3 instances of service *A* in the composite service graph" cannot be evaluated in an incremental fashion with the current approach and can only be checked post-composition. In general, while our current system allows for the evaluation of constraints that include the use of certain aggregation functions such as maximum and average, a more thorough analysis that includes this feature is needed.

Second, the current service composition process considers service instances to include in the composite service graph only if they pass the policy checks that are applicable to them and skips those that fail. However, one can improve the service selection process by making use of the fact that policies have failed so as to guide the selection of the next service instance to be considered in a more targeted manner, as opposed to the current blind iteration of the candidate service instance set.

Last, while the evaluation of our system so far has tried to establish the simple, baseline performance with policies that refer to individual service instances, a more realistic experiment scenario would include more complicated policies that are applicable to properties of the complete composite service graph. This would also allow one to assess the impact of policy checking in the overall service composition execution time in the case where *graph properties* have to be computed so as to then check constraints. We place this item high on our priority list for further investigation.

VIII. Related Work

Research community working on distributed computing and service oriented computing (SOA) has widely studied service composition which has evolved as work flow representation and business process. The access control and policy driven mechanisms for web-services are well-studied and researchers have proposed promising mechanisms. However, unlike web services, service composition in WSNs is an emerging area and especially policy driven dynamic service composition is yet to be explored. The protocols devised for web service composition in SOA are computationally sound for processor and memory rich systems, but are not readily applicable to resource constrained WSNs environment. Here we present a short review of the work done on secure and policy based web-service composition.

A broker based architecture to compose services is proposed in [3]. To address security requirements (namely security capabilities) of the services, these capabilities are validated before final composition. Moreover they assume the existence of one or more trusted entities which can make sure that the services are checked against the constraints before final composition is performed. In [4], the authors propose a security wrapper based approach to ensure that the security policies are respected in service composition. Their approach requires that the services taking part in composition to use semantically equivalent construction. The security wrapper around service is automatically generated from policies and it implements checks and cryptographic operations needed to ensure the feasibility of composition.

In [14], the authors propose mechanism for information flow control in service chains. The authors propose a three-phase mechanism to improve the efficiency of composition by filtering out candidate services which are unlikely to satisfy composition constraints. Like some other approaches, the authors in [14] also propose protocol which uses remote services for checking and validation of policies. After the appropriate services are found (i.e., those that adhere to requirements as well as to the flow control policies of the requester), the service composer builds up the service chains.

A negotiation based technique for secure service composition is proposed in [10]. The authors introduce framework for security-aware service composition which establishes contracts between various services by negotiating security properties. In their approach, the authors represent security properties and their combinations in a tree-like structure that has nodes divided into groups based on preferences. Iterative bi-lateral negotiations are performed to come up with agreed contract for service composition. This contract is then validated against low-level security requirements to ensure secure service composition.

An access control mechanism for web service composition is proposed in [21]. In the proposed mechanism, any request for service invocation is first validated against a policy file maintained for each service. Moreover, this paper proposes a composite method in which all policy files related to elementary web services are combined into an integrated file based on which the access control mechanism is defined for composition. In [5] and [6], an ontology based approach for policy based web service composition is proposed. The authors achieve service composition by forming service flow for which rules are stored in knowledge database and services are represented as topic ontology. Policies are implemented in form of syntactic and semantic rules. In [16], the authors propose an access mechanism for regulating access to data in the distributed databases in a way similar to database privileges mechanism that is used in relational databases. In their approach, the authors test the access and authorization of queries that are run on distributed databases. Queries attempting to access unauthorized data are suppressed by the system from proceeding.

A policy driven approach to transform business policies to low-level service policies is proposed in [17]. The service policies are maintained in a centralized repository from where they are distributed to related local repositories. The authors distinguish between group level policies and service level policies both of which are enforced at run-time. In [15] the

authors propose a Hierarchical Task Network (HTN) based approach for web service composition in which a task network hierarchically abstracts the service composition. The authors use OWL ontologies to describe composition template which is then converted to HTNs. A rule-based constraint policy language is suggested in [18] for providing enforcement of policies on top of the composition graph. The basic idea presented in the paper is to convert policies and the composition to an ontology. Such an approach requires that the system performs the full composition first and then checks it against policies, which can degrade the system performance. A UML based approach to syntactically define service composition and policies is suggested in [13]. It uses activity diagrams and other transformation techniques to address issue of service composition.

IX. Conclusion

Due to their distributed method of deployment, sensor network applications naturally fit a composition paradigm that interconnects multiple, light-weight aggregation and transformation data processing services that run on individual sensor nodes with one another, to develop a complete data collection and analysis application. As with any networked application, it is subject to resource management and security constraints that are expressed through policies. In this work, we propose a policy-aware sensor service composition framework that can automate the process of combining simple sensor services into larger ones, while, at the same time, complying with pre-specified policies. We also present an implementation of our framework in the Sensor Fabric, a middleware for developing sensor network applications and present preliminary performance evaluation results of our prototype. Our system allows for the evaluation of attribute-based policies on service composition graphs, signifying a departure from the traditional focus on access control policies of prior proposals. Our ongoing work is focused on supporting aggregation functions for the evaluation of policy constraints, as well as conducting a more thorough performance evaluation of our initial prototype, on a real sensor testbed.

X. Acknowledgment

This research was sponsored by the $U.S.$ Army Research Laboratory (ARL) and the $U.K.$ 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 author(s) and should not be interpreted as representing the official policies, either expressed or implied, of the $U.S.$ ARL, the $U.S.$ Government, the $U.K.$ Ministry of Defence or the $U.K.$ Government. The $U.S.$ and $U.K.$ Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

References

[1] G. Bent, P. Dantressangle, D. Vyvyan, A. Mowshowitz, and V. Mitsou. A dynamic distributed federated database. In *In Proc. of ACITA*, Sep 2008.

[2] E. Bertino, A. Squicciarini, and D. Mevi. A fine-grained access control model for web services. In *In Proc. of Services Computing Conference (SCC)*, 2004.

[3] B. Carminati, E. Ferrari, and P. Hung. Security conscious web service composition. In *Web Services, 2006. ICWS'06. International Conference on*, pages 489–496. IEEE, 2006.

[4] Y. Chevalier, M. Mekki, and M. Rusinowitch. Automatic composition of services with security policies. In *Services-Part I, 2008. IEEE Congress on*, pages 529–537. IEEE, 2008.

[5] S. Chun, V. Atluri, and N. Adam. Policy-based web service composition. In *Research Issues on Data Engineering: Web Services for e-Commerce and e-Government Applications, 2004. Proceedings. 14th International Workshop on*, pages 85–92. IEEE, 2004.

[6] S. Chun, V. Atluri, and N. Adam. Using semantics for policy-based web service composition. *Distributed and Parallel Databases*, 18(1):37–64, 2005.

[7] E. Damiani, S. D. C. di Vimercati, S. Paraboschi, and P. Samarai. Fine grained access control for SOAP e-services. In *In Proc. of International Conference of Web Services (ICWS)*, 2001.

[8] DMTF. Common Information Model. http://dmtf.org/standards/cim.

[9] S. Geyik, B. Szymanski, P. Zerfos, and D. Verma. Dynamic composition of services in sensor networks. *IEEE International conference on service computing SCC*, pages 242–249, 2010.

[10] J. Han, R. Kowalczyk, and K. Khan. Security-oriented service composition and evolution. In *Software Engineering Conference, 2006. APSEC 2006. 13th Asia Pacific*, pages 71–78. IEEE, 2006.

[11] J. Ibbotson, S. Chapman, and B. K. Szymanski. The case for an agile SOA. *First Annual Conference of the International Alliance*, 2007.

[12] IBM Research. Policy Management Library. https://www.ibm.com/developerworks/community/groups/service/html/communityview?communityUuid=ed556565-1d91-4289-94ae-213df1340350.

[13] J. Rossebø and R. Runde. Specifying service composition using uml 2. x and composition policies. *Model Driven Engineering Languages and Systems*, pages 520–536, 2008.

[14] W. She, I.-L. Yen, B. Thuraisingham, and E. Bertino. Policy-driven service composition with information flow control. *IEEE International Conference on Web Services*, 2010.

[15] S. Sohrabi and S. McIlraith. Optimizing web service composition while enforcing regulations. *The Semantic Web-ISWC 2009*, pages 601–617, 2009.

[16] S. Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Controlled information sharing in collaborative distributed query processing. In *Distributed Computing Systems, 2008. ICDCS'08. The 28th International Conference on*, pages 303–310. IEEE, 2008.

[17] S. Wang and M. Capretz. A policy driven approach for service-oriented business rule management. In *Industrial Informatics, 2007 5th IEEE International Conference on*, volume 2, pages 713–718. IEEE, 2007.

[18] W. Wei and T. Yu. The design and enforcement of a rule-based constraint policy language for service composition. In *Social Computing (SocialCom), 2010 IEEE Second International Conference on*, pages 873–880. IEEE, 2010.

[19] J. Wright, C. Gibson, F. Bergamaschi, K. Marcus, R. Pressley, G. Verma, and G. Whipps. A dynamic infrastructure for interconnecting disparate ISR/ISTAR assets (the ITA Sensor Fabric). *IEEE/ISIF Fusion Conference*, 2009.

[20] N. Xu. A survey of sensor network applications. *IEEE Communications Magazine*, 40, 2002.

[21] J. Zhu, Y. Zhou, and W. Tong. Access control on the composition of web services. In *Next Generation Web Services Practices, 2006. NWeSP 2006. International Conference on*, pages 89–93. IEEE, 2006.