9th International Conference on Service Computing, IEEE SCC 2012, June 24-29 2012,
Hyatt Regency Waikiki Resort and Spa, Honolulu, Hawaii, USA

# Utilizing PCFGs for Modeling and Learning Service Compositions in Sensor Networks

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

***Abstract*— Service composition in sensor networks combines elementary services with a specific functionality to create a service with higher level functionality. The previous efforts in automating composition were sending full information about all services across the entire sensor network, creating a security risk and imposing significant communication overhead. Furthermore, learning based composition or error detection methods do not consider global information, leading to inefficiencies in the generated composition graphs. In this paper, we propose a probabilistic context-free grammar (PCFG) based modeling technique to construct service compositions. The successful compositions created for the given application are treated as statements belonging to an efficient composition PCFG of this application. The given set of such compositions is used to derive this PCFG automatically. Future composition could be then easily constructed with the help of such PCFG. We present our methodology for achieving such modeling and provide examples of its use to demonstrate its advantage over previous work. We also evaluate the resulting improvements in performance of compositions and in the costs of their creation.**

***Index Terms*— sensor networks; service composition; PCFGs;**

## I. Introduction

Service oriented architecture (SOA) for wireless sensor networks (WSN) have recently been proposed as a means to abstract the sensor network as a set of services (programs integrated within sensor nodes) that provide a certain functionality. In this context, the task of service composition is to efficiently combine a set of services so that a more complex functionality can be realized. In our initial work for SOA for WSN [1], we showed how to represent a composition as a graph of connection of services and how to automatically combine services in an efficient way. While such an automated scheme improves energy efficiency of the solution, during composition creation this scheme suffers from communication overhead and a security problem because metadata about services are sent between all potential services. Since the metada may include mission critical information, in domains such as military applications, the user may not want such information to be moved around freely. Manual connection of services by the user is a possible solution to this problem that has been followed in the literature. However, such a method becomes error-prone with the increasing composition size.

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.

In this work, we are addressing the discussed above problems of security, high overhead and performance involved in service composition in sensor networks. To be able to create high performance service compositions with low overhead, we propose to use probabilistic context free grammars (PCFGs) to learn the composition schemes in a sensor network.

Informally, PCFGs are regular context free grammars in which production rules are assigned probabilities reflecting their usage frequency. We assume that the efficient initial compositions of services are available, for example by creating compositions, evaluating their performance and collecting the most efficient one. They are used to create a PCFG such that each initial efficient composition for the given network is a sentence belonging to the language defined by this grammar. We also provide a method on how to generate a string representing a service composition. Each PCFG constructed via our method is specific to a single sensor network instance with predefined set of elementary services and possible information flows between them. In such a PCFG, a production rule with high probability assigned to it represents a highly efficient subcomposition step. The hidden assumption here is that the frequently used service combinations appear in the intial set of efficient compositions for a reason, because they are more advantageous than the alternatives. Hence, the constructed PCFG defines a *language* for efficient service compositions. The contributions of the paper can be summarized as:

- A PCFG and its corresponding language to represent efficient service compositions as strings in this language,
- A methodology for generating a PCFG from previous compositions,
- A method for finding frequently used subcomposition patterns in PCFG sentences, improving efficiency of the service design,
- A method to utilize the PCFGs for creating efficient compositions and reducing the commmunication and computational overhead of this process.

The rest of the paper is organized as follows. Next, we provide background on the notion of service composition and explain the language for describing a service composition as a string, which basically is a way to represent a service composition graph. We continue with a background on PCFGs and our utilized method of PCFG construction in [2]. Then, in Section III,we discuss the basics of our methodology. We provide a way of finding subcompositions while constructing a PCFG. We also explain why generating PCFGs is a better solution than just statistically inferring the service connection probabilities, or simply examining one level service usage

sequences. We also discuss how a composition is created from the constructed PCFG as the system runs. In Section IV, we compare the PCFG-based composition methodology to our previously presented automated composition technique [1] in terms of performance, as well as the previously examined modeling/learning techniques for this application domain. Next, we talk about previous work on service composition learning techniques in Section V. We finalize the paper with our conclusions and future work in Section VI.

## II. Background

### A. Service Composition in Sensor Networks

Service oriented architecture for sensor networks view any application in a sensor network as a collection of component services assembled in a data flow graph that describes the composite service. In [1], we have proposed a service model for sensor networks, in which each service is defined by the set of inputs (information) on which this service operates, the set of outputs that it produces, the function that maps these inputs to outputs, as well as the *metadata* that provides the properties of the service (metadata can contain virtually anything, e.g. cost of activating the service, such as delay, energy needed, security level, and sensor specific properties, such as its *location*, *precision*, and *reliability*).

Based on this model, the *service graph* can be defined as the set of vertices representing the services, and the set of directed edges between the vertices representing the possible information flows between those services. To fulfill the requirements of the desired service, the *service selection* process is used to choose the subset of possible input providers (other ends of the *incoming* edges in the service graph) that ensures the correctness of the composition. The *service composition* problem, as defined in our study however, is the choice of the services (vertices in a service graph) and information flows (edges in a service graph) so that a *user request* can be produced by this composed service with the lowest cost. In the heuristics proposed in [1], (namely *top-down* and *bottom-up* approaches), the cost estimates are used at each level of the algorithm. Therefore, to satisfy its inputs, a service chooses a subset of its possible input providers based on estimates of their costs.

### B. Describing Service Compositions as Strings

As aforementioned, we define service composition as a data flow graph where the directed edges represent the input being provided from the service at the start of the edge to the service at the end of the edge. To describe a service composition as a string, we place service names within the parentheses to represent the services used as input providers. Hence, the following grammar for the language to desribe service compositions can be used, assuming that there are $n$ elementary services defined in the given sensor network:

<composition> → <service_name>(<composition_list>) |

<service_name>

<composition_list> → <composition> |

<composition>, <composition_list>

<service_name> → service$_1$ | service$_2$ | ... | service$_n$

Hence, each node in the data flow graph is represented in the string by its name followed by the list, enclosed within parentheses, of nodes in the subgraph rooted in it. In the above graph grammar, we have just four tokens: *left and right parantheses* which encapsulate the set of services used by a service whose name preceeds the matching parantheses, *commas* to separate compositions, and *service names*. Such a grammar of course can only represent service compositions which are acyclic (i.e. the composition string is finite).

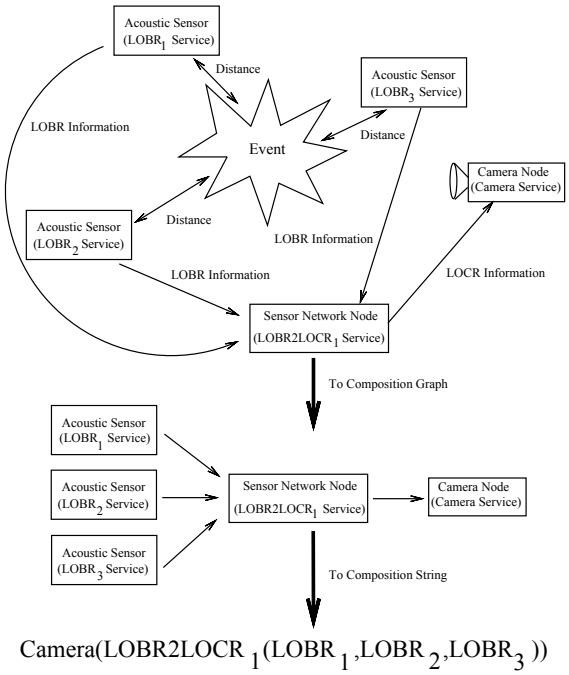



**Fig. 1: An Example to Illustrate the Service Composition Language**

To give an example, we use the application from [3] which requires the composite *Camera* service. In Figure 1, the services $LOBR$ (with indices to distinguish them) give a measurement ($LOBR$ stands for Line of Bearing Report) of the acoustic sensor's distance from a source of sound regarded as an event. Later, these measurements from three acoustic sensors are sent to the $LOBR2LOCR$ service which uses triangulation to detect the location of the event ($LOCR$ stands for Location Report), and sends this information to the camera sensor (another service), which then starts monitoring the event. The $LOBR$ services require no input, so they use the "<composition> → <service_name>" rule of the language given above for their composition. However, $LOBR2LOCR$ uses services $LOBR_1$, $LOBR_2$ and $LOBR_3$, hence its composition is like a function call, which starts with the service's own name ($LOBR2LOCR$) and is followed by the set of

services (with their corresponding composition strings) that are used to satisfy its inputs. Finally, the service $Camera$ uses composite $LOBR2LOCR$ service, so its composition string contains both $LOBR2LOCR$ and its composition subgraph.

A simple parser can extract the links from the given composition. Furthermore, as mentioned in the introduction, our methodology uses the strings of the previously run compositions to encompass in the PCFG the rules of combining services efficiently. Thus, the constructed grammar can be used to either create efficient compositions automatically, or to check the validity of the compositions for correctness and efficiency.

### C. Probabilistic Context Free Grammars (PCFG)

A probabilistic context free grammar consists of five parts: ($i$) a list of terminal symbols, $S_t$, that can be present in a sentence generated by this grammar, ($ii$) a list of nonterminal symbols, $S_{nt}$, each representing a set of strings of nonterminal or terminal symbols, ($iii$) *Start* nonterminal from which generation of every sentence of this grammar starts, ($iv$) set of rules ($R$) which define precisely what strings a nonterminal can produce, and ($v$) probabilities of using rules ($Pr$) to define how frequently a rule is to be used when generating a string from the corresponding nonterminal.

Let's provide a simple PCFG based on the example from Figure 1. For disambiguation, we are representing nonterminals with uppercase letters and terminals with lowercase letters. For our purposes, let's assume the camera service can be composed in two ways: "camera ( lobr2locr$_1$ ( lobr$_1$ , lobr$_2$ , lobr$_3$ ))" with 0.8 probability, and "camera ( lobr2locr$_1$ ( lobr$_4$ , lobr$_5$ , lobr$_6$ ))" with 0.2 probability. Then a PCFG corresponding to such a case can be provided as below:

Start $\rightarrow$ camera( LOBR2LOCR ) (1.0)

LOBR2LOCR $\rightarrow$ lobr2locr$_1$( lobr$_1$ , lobr$_2$ , lobr$_3$ ) (0.8) |
lobr2locr$_1$( lobr$_4$ , lobr$_5$ , lobr$_6$ ) (0.2)

In the above grammar, *Start* and *LOBR2LOCR* are nonterminals, the numbers in parentheses are the usage probabilities of the rules that they follow, and finally, *'lobr2locr$_1$'*, *'lobr$_1$'* ... *'lobr$_6$'*, *'camera'*, *'('*, *')'* and *','* are terminal symbols.

In our previous work [2], we have presented an efficient and fast PCFG construction algorithm that takes as input the set of sentences that are generated by the unknown grammar and constructs the most likely concise grammar with the matching rule probabilities. We have utilized this scheme for event recognition in sensor networks.

The construction method consists of two parts: ($i$) Sample Incorporation, and ($ii$) Application of Operators. The first part constructs the initial grammar capable of generating sentences from the training data ($D$) and only such sentences. In the second step, two operators, *merge* and *chunk*, are applied to the initial grammar to generalize it and make it shorter. The grammar at each step of operator application is evaluated by its Bayesian *a posteriori* probability, and the search for where to apply an operator to the current grammar is guided by this probability. The details are described in [2].

## III. Modeling Service Composition as a PCFG

In this section, we provide the details of how the PCFG can encapsulate service composition information and patterns specific to a sensor network instance with predefined set of services integrated on the sensors. Based on the way to describe service compositions as strings (Section II-B), we describe how a PCFG that describes a language for the efficient compositions can be constructed from a set of strings representing previous compositions. We also decribe how frequent subcompositions (or composition alternatives) can be inferred during the PCFG construction and how to use them to improve the service composition construction.

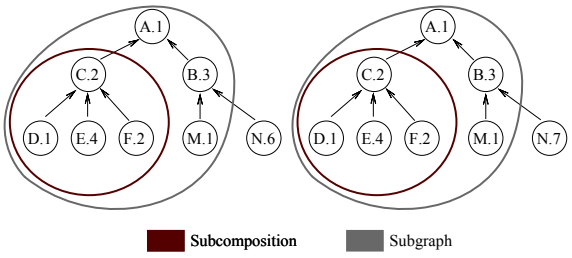
Although there is a lack of previous efforts on learning the service compositions in sensor networks or web applications, a similar problem of learning in component based systems has been widely studied in recognizing the causes for system errors. The most relevant paper on error detection that also utilizes PCFGs, was written by Kiciman et al. [4]. It presents a system called *Pinpoint* that detects faults in the application layer of internet services. The detection algorithm constantly monitors software component interactions. The training stage of the system provides Pinpoint with steady-state behavior of the system while in the run-time, faults are recognized due to behavior that is too divergent from the general case. The first of the two main methods used in Pinpoint is the weighted graph that models the frequency of the component (service) interactions in the system (each component is a vertex in the weighted graph). The second method uses PCFGs to model order in which other services are used. Hence, every service is a nonterminal that points to an ordered set of other services. In this section, we also give a simple example of advantages resulting from our methodology compared to [4]. This is to show that globally available composition information for the system allows for generation of PCFGs with better composition capabilities.

We finalize the section with the method of regenerating service compositions according to the availability of services during the application lifetime, and what advantages such regeneration provides compared to our previous efforts on automated composition [1].

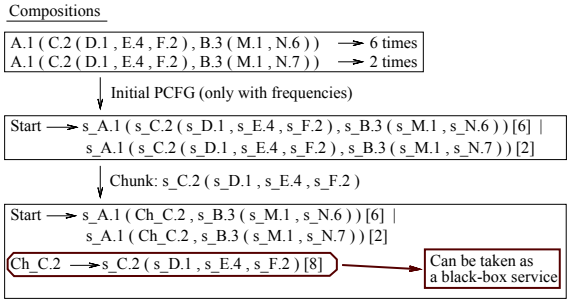
### A. Finding Subcomposition Patterns via PCFG Inference

In our previous work [2], we have adopted the usage of a *chunk* operator as a means of shortening the constructed grammar. This operator basically looks for frequently appearing patterns in the grammar and replaces them with a new nonterminal, which in turn generates this pattern. In this paper, we are using similar methodology to find subcomposition patterns which are used frequently, so in practice, it would be advantageous to encapsulate them as black-box services.

A subcomposition is a subset of the connections and services that are utilized for a composition. Furthermore, we assume that any subcomposition describes how an intermediate service is composed down to the source services (i.e. services that do not require any input from other services) and is complete (i.e. all inputs and outputs are satisfied). This is




(a) Subcomposition Example




(b) Subcomposition Discovery by the Chunk Operator

**Fig. 2: Finding Subcompositions via PCFG Inference**

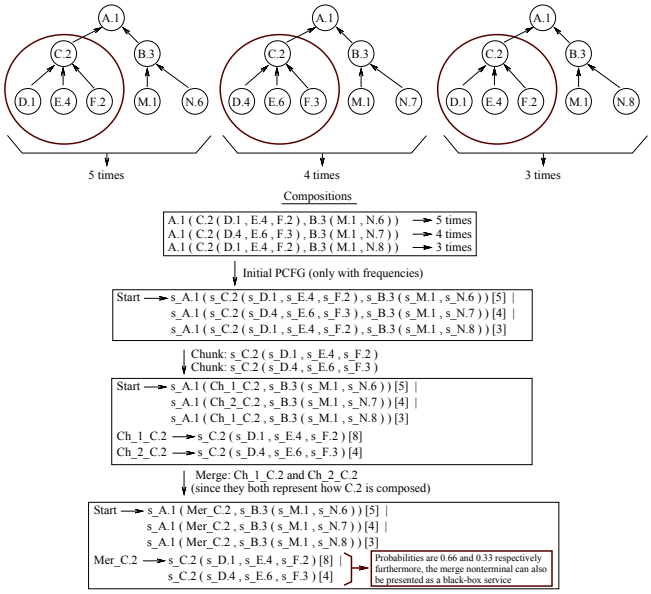



**Fig. 3: Subcomposition Example**

the main difference of the frequent subcomposition inference from *graph mining*. We do not find a frequent subgraph of the previously seen composition graphs, but rather we find a frequent composition scheme of one of the services used in previously seen composition graphs (down to services that do not require any input from other services to execute). The example in Figure 2a shows two types of frequent structures that have been marked in two composition schemes for the service *A.1*. Elevating a subgraph as a subcomposition depends on its usage frequency.

In the previous section, we have described how a composition scheme can be represented as a parenthesized string. In such a language, a subcomposition consists of all the services (hence further subcompositions too) within two matching parentheses. Such a subcomposition describes how the service name preceding the matching parentheses is composed by a set of other services. During the PCFG inference from the set of compositions, we check for the frequency of each subcomposition and compare it to a threshold before assigning to it a nonterminal representing a new black-box service. This threshold is application specific, and depends on the size of the training data. Figure 2b shows an example of such a replacement, where the example from Figure 2a is presented as a PCFG. The numbers in the brackets show how many times the composition has been used. The subcomposition shown in Figure 2a is assigned the chunk nonterminal (*Ch_C.2*), which can later be advertised in the system as a black-box service.

Once the subcompositions are found, the probabilities of alternative subcompositions can be found by *merge* operator [2] in PCFG inference. Merge operator combines the rules of two nonterminals into a new nonterminal, and replaces the occurrences of both these nonterminals with the newly constructed nonterminal. During the PCFG construction, this operator combines only nonterminals that are the subcompositions of the same service. Consider the example in Figure 3. In this case, two frequent subcompositions for the service *C.2* are combined into a new nonterminal, which can be provided later to the end-user as a single black-box service.

The advantage of the process of finding subcompositions and the alternatives for such subcompositions are two-fold. First, the efficiency of composing services improves. Being able to detect efficient compositions from previous requests for services (hence never directly composing them) saves processing time [1]. Second, the service space is reduced because certain elementary services which are used only in frequent subcompositions can be removed.

### B. Advantage of the PCFG-based Service Composition

An example in Figure 4 illustrates why training a PCFG on service compositions yields better results than the weighted interactions of Kiciman et al. [4]. In the figure, rectangles represent services and the circles represent the sensor nodes while the links between the circles represent the logical communication connections. In the example, the service $C_1$ uses two types of services to receive its required inputs: an instance of $A$, and an instance of $B$, each indexed to distinguish between their different instances. In this sample network, if $A_1$ and $B_1$, or $A_2$ and $B_2$ are chosen together, then their data flows meet at two nodes (Node_5 and Node_6, respectively), causing congestion. Hence, the efficient compositions use either $A_1$ and $B_2$, or $A_2$ and $B_1$ together to provide the inputs to $C_1$. PCFG will store and utilize this pattern. However, approach similar to [4] that stores the edge weights will assign equal usage probabilities to $A_1$, $A_2$, $B_1$ and $B_2$. Consequently, the usage of pair $(A_1, B_2)$ has the same probability (0.25) as of

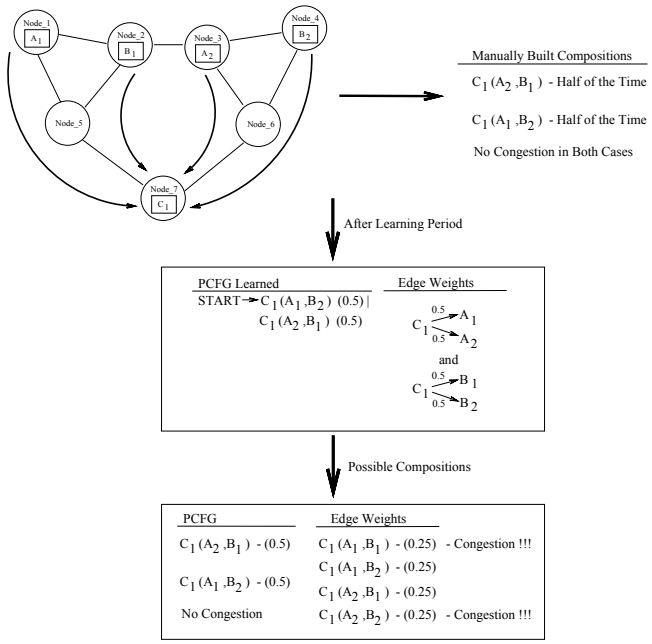



**Fig. 4: An Example to Present the Advantage of the PCFG-based Service Composition over Edge Weights in [4]**

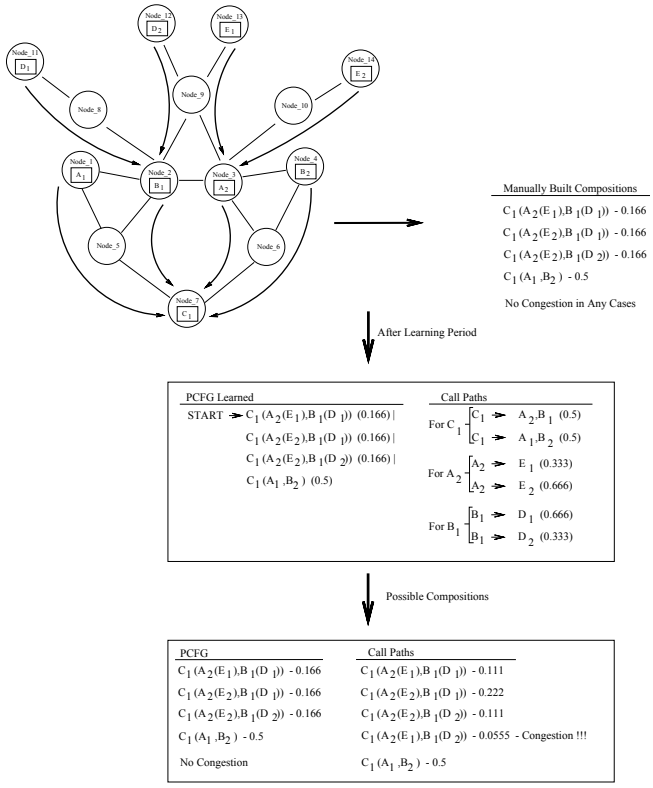



**Fig. 5: An Example to Present the Advantage of the PCFG-based Service Composition over Call Paths in [4]**

pair $(A_2, B_2)$ even though the latter causes the congestion, while the former does not.

The remedy proposed by Kiciman [4], is to use the list of other components that are called by a component. Although this remedy works for the given example, it fails with minimal extension of the example shown in Figure 5 where services $A_2$ and $B_1$ are able to utilize the services $D_1$ or $D_2$ and $E_1$ or $E_2$, respectively. Since the call paths are only examined at a local level, the remedy will choose a composition with $A_2$ using $E_1$ and $B_1$ using $D_2$ at the same time (hence causing a congestion at $Node\_9$, since their data flow routes intersect there). However, we train PCFGs from entire compositions, so our method knows that in no efficient composition $E_1$ and $D_2$ have ever been used together.

In the previous subsection, we have presented how the *chunk* and *merge* operators help with the discovery of subcompositions and the alternatives for a frequent subcomposition. It can easily be argued that such a process will suffer from the same myopic problem of creating suboptimal compositions from which the *edge weights* and *call paths* methodology discussed above suffer. We hereby present that this problem can be avoided by using an entropy-based scheme for applying these operators. Please note that the chunk operator as given in this paper finds only the frequent subcompositions, hence has no generalizing effect, and will not create suboptimal compositions. However, doing the merge may cause such an effect, hence should be used with great care.

Suppose we have found $n$ subcompositions for the same service $S$, which have frequencies $f_1 \dots f_n$, and the sum of these frequencies is $\sum_{i=1}^{n} f_i = f_{sum}$. When we combine all these alternative subcompositions into a single new subcomposition nonterminal (by the merge operator), the probability for each subcomposition in this nonterminal (i.e., the corresponding rule probability) becomes $p_i = \frac{f_i}{f_{sum}}$. In this setting, the total probability of a subcomposition being used in place of another subcomposition (i.e. the probability of generalized usage) is "$\sum_{i=1}^{n}$[(Probability of generating a case where $i$ was used) $\times$ (Probability of using subcomposition other than $i$)]"=$\sum_{i=1}^{n} p_i(1-p_i) = \sum_{i=1}^{n} p_i - p_i^2 = 1 - \sum_{i=1}^{n} p_i^2$. This value is the total probability that the PCFG-based service composition may use subcomposition $j \neq i$ when in reality $i$ would be used if no generalization had occurred. We want such generalization to have as low probability as possible, hence $\sum_{i=1}^{n} p_i^2$ to be as large as possible. This gives us a bound on which frequent subcompositions are more desirable, and which merges are more beneficial to hold generalization probabilities low while giving the advantage of providing black-box services to the end-user. Basically, we want to find very frequent subcompositions, but we also want subcomposition alternatives where one dominates the others in terms of usage frequency (which increases the value of $\sum_{i=1}^{n} p_i^2$). We evaluate such an approach in Section IV.

### C. Utilization of PCFGs to Generate Compositions

Once a PCFG is constructed from previous composition examples, the next step is to utilize it to generate compositions.

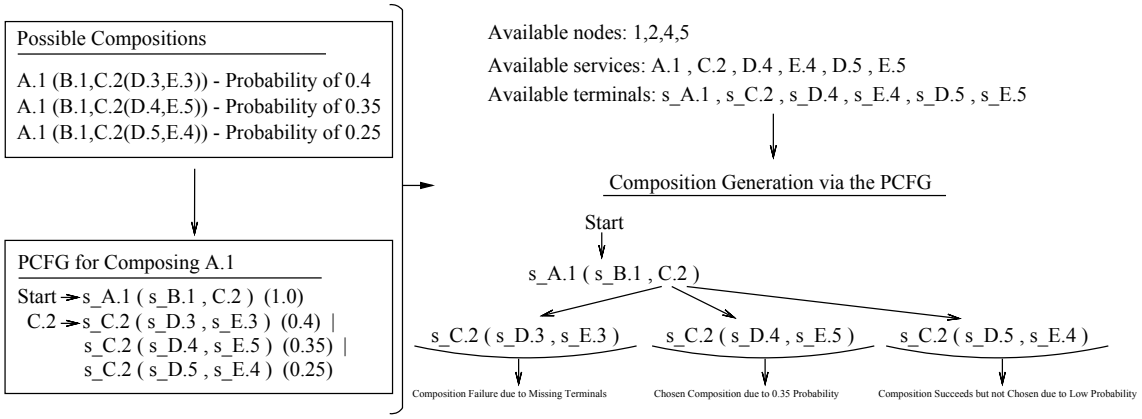



**Fig. 6: An Example on How to Generate a Composition via the PCFGs Given the Set of Available Services**

For this purpose, we propose the usage of a centralized scheme where the PCFG is held at a centralized decision maker, where the list of available services are also kept. The service name together with the id of a sensor node on which it resides constitute a unique terminal symbol for our PCFG. We furthermore assume that the high probability rules represent the fact that certain composition schemes have been preferred over others, hence denote a more efficient composition. We leave the PCFGs where rules are labeled with performance values for future work.

The creation of a composition essentially involves generating a sentence from the PCFG, where the productions with higher probability are kept, and the productions with unavailable services are directly eliminated. In Figure 6, we give an example on how the composition is generated at the centralized decision maker. In this example, there are five nodes and five basic services (A, B, C, D and E). We denote an instance of a service at a specific node with a dot, e.g. D.5 means an instance of service D residing on node 5. Each such instance is made into a terminal in the presented grammar with the '*s_*' in front of it, to distinguish between terminals and nonterminals. As it can be seen, although the composition with probability $0.4$ is more desirable (due to previous usage frequency, also represented in the grammar), we cannot create it since node 3 and all of the services on it are unavailable, and the creation of such composition from the grammar with the related terminals cannot complete. Instead, the composition with 0.35 probability is selected. Please note that nodes need to exchange messages to update the list of available nodes, as described in [1]. The benefit of using the PCFGs here are three-fold. First, we already have the probabilities of use of each service in the past that represent the preference for using them currently. Second, the exchange of messages between services with low preferences can be eliminated. Finally, metadata information can be limited to indication if a node is available; we do not need all input/output lists since we already know the links between services. This is a significant reduction compared to an input/output matching based automated composition method, hence improvement of both communication overhead and security of composition construction.

## IV. Evaluation

In this section, we demonstrate two advantages of the PCFG-based composition. First, we will show that by changing the lower bound on the generalizing threshold (as defined in Section III-B), the number of the composition alternatives that are substituted in the created composition changes. More precisely, the lower the generalizing threshold, the more likely it is that the optimal subcomposition scheme will be replaced with another one. From that point of view, the methods of *Edge Weights* and *Call Paths* can be seen as generalizing without any such threshold. We present the effect of the value of generalizing threshold in comparison to *Call Paths* methodology in which generalizing is a bit more suppressed in general.

We also evaluate how the PCFG-based composition helps in reducing processing during the composition creation, as compared to an automated composition scheme from [1]. We have already discussed how the identification of subcomposition patterns creates black-box services for the end user. We evaluate how the generalization threshold affects the faster composition (since the lower the generalization threshold, the more black-box services there are and the less time it takes to compose the solution).

### A. Simulation Setting

For our evaluations, we have implemented a simulation as a simple version of the application in Figure 1. See Figure 7 for a detailed presentation of this scenario. We have a 5x5 grid where each 2x2 subgrid is in the range of a camera sensor, hence covering an area of four grid rectangles. Each camera connects to three acoustic sensors (one from each rectangle, hence no acoustic sensor is chosen from one rectangle) to detect the location of a potential target. There are 16 2x2 subgrids, hence 16 cameras, and each subgrid rectangle (there are 25 of them) has three acoustic sensors (75 in total) for cameras to choose from.

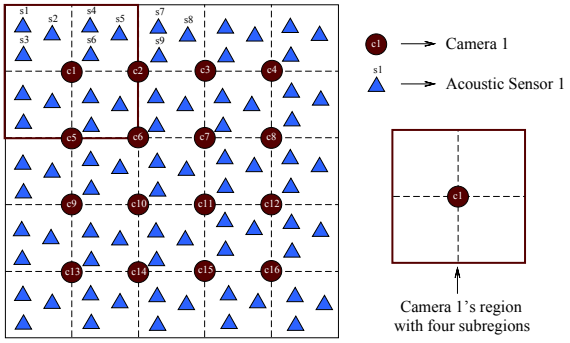



**Fig. 7: Simulation Application**

Each user request in our simulation is for 2 to 4 camera service's composition, while the cameras have a preset probability for using each acoustic sensor in each subregion. We have set the most likely acoustic sensor in a subregion with probability between 0.85 and 0.95 ($p_1$) for each camera (i.e., each camera separately preassigns a likelihood to each acoustic sensor in each subregion). The other two sensors are assigned probabilities from the remaining probability range (with total probability of $p_2$+$p_3$=1-$p_1$). We assume the processing cost for choosing an acoustic sensor (selection process) in a subregion to be three units for an automated composition scheme. For the PCFG however, we take the number of symbols seen at the first level from the *START* nonterminal, i.e. each black-box service is only given a processing cost of one.

### B. Results

In our simulations, we used 10 cases of the simulation scenario given in the previous subsection. Each of these cases contains 1000 user requests, which chooses 2-4 camera services to compose. Later, we use the outputs of these simulations, and generate PCFGs with our proposed method. We changed the allowance of generalization for service's subcomposition to be chosen as a black-box, as given in Section III-B. We have asserted in our experiments that each service should occur at least 20 times in the training data to be chosen as a parameter to the *chunk* or *merge* operator, regardless of its generalization threshold. This threshold is application specific, and is kept constant in the experiments since it is not worth representing a service as a black-box abstraction if it is barely used in the application.

A sample of the PCFG rules constructed from the output of our simulations is presented in Figure 8. In the figure, the *START* rule produces a user request from the cameras 3 (*c3*) and 12 (*c12*), i.e. *u_3_12*. This request uses the black-box representations of camera services 3 and 12, which are represented as nonterminals constructed by the merge operator as previously demonstrated (i.e. *M_c3* and *M_c12* where *M* represents the merge). Although the black-boxes include more rules in the exact output than presented here, we have shown the highest four, and normalized them to reach a sum of probabilities equal to 1.0, again for presentational purposes. In the rules, a camera service uses three acoustic sensors (*s* followed by the id, as shown in Figure 7); this is represented as a string according to the composition representation language (see Section II-B).

```
START -> u_3_12 ( M_c3 , M_c12 ) (0.005)
M_c3 -> c3 ( s7 , s11 , s27 )  (0.26)  |
        c3 ( s7 , s22 , s27 )  (0.23)  |
        c3 ( s11 , s22 , s27 ) (0.21)  |
        c3 ( s7 , s11 , s22 )  (0.30)
M_c12 -> c12 ( s45 , s56 , s59 ) (0.31) |
         c12 ( s40 , s45 , s59 ) (0.22) |
         c12 ( s40 , s56 , s59 ) (0.22) |
         c12 ( s40 , s45 , s56 ) (0.25)
```

**Fig. 8: A Sample from the Constructed PCFGs**

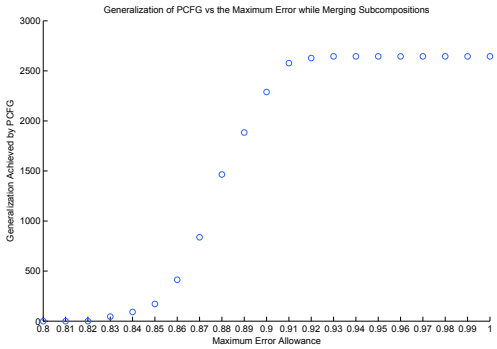



**Fig. 9: Generalization Results**

Figure 9 presents the generalization achieved on the training data by varying the allowed generalization metric (1-*generalization threshold*). This metric can be at most 1.0, meaning there are infinitely many rules each with probability going to 0 (e.g. $1-\sum_{i=1}^{\infty} p_i^2 = 1$). We present the results with different generalization allowance in range [0.8, 1.0], since we have found out that there are no possible merges until 0.83 (this can be observed from the figure since generalization is 0 until x-axis is 0.83). The y-axis in the figure represents how many subcompositions in training data would be substituted with an alternative one, and increases with the generalization allowance. The method of *Call-Paths* [4] is the point where the allowance is 1.0 (i.e. all generalizations are allowed, hence every service subcomposition is a black-box). The automated composition [1] is the one with the generalization allowance of 0.0, hence results in the generalization of 0.

In Figure 10, we provide the results showing the composition creation processing cost improvements obtained by the PCFG-based composition using black-box service descriptions as a function of the generalization threshold. As mentioned before, increased generalization threshold allow more merges that combine more alternative service subcompositions into a single nonterminal (black-box representation). We have assumed that the automated subcomposition chooses one of the three acoustic sensors for each subregion when a camera

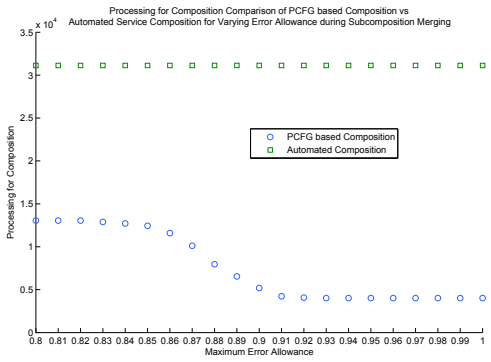



**Fig. 10: Processing Cost Results**

service is activated, hence brings a three-unit processing cost (for each acoustic service). In contrast, when the PCFG creates a black-box representation for this camera service, the corresponding cost incurred is just one-unit. The figure clearly shows that the increased generalized threshold helps with lowering processing cost of composition. Without the generalization (i.e., when generalization threshold is less or equal to 0.83 and all merges are suppressed), the PCFG method works only through its rules, and hence brings down the cost of choosing each camera or acoustic service to one (for acoustic services, the automated composition incurs the cost of three-units since we assume an evaluation of selection the best acoustic sensor among the three possible ones in the given subregion). Furthermore, the method of *Call-Paths* [4] is represented by the point where the x-axis has 1.0 allowance, hence every camera service is made into a black-box representation of its subcomposition.

The results demonstrate that our PCFG-based composition methodology lowers processing cost for creating compositions, and the fine tuning of the application specific generalization threshold can resolve a trade-off between processing cost and composition generalization.

## V. Related Work in Service Composition Learning

As already mentioned, we are not aware of any previous efforts on learning the service compositions in sensor networks or web applications as an automated composition generation method by learning from previous compositions. A similar problem of learning in component based systems, however, has been widely studied in systems recognizing the causes for system errors (see surveys in [5] and [6]). We have already discussed an important work [4], in Sections III and IV, which we also use for evaluating our approach. We now review similar studies within the same domain.

The authors of [7] introduce *Gingko*, a mechanism which allows users to correlate *causal paths* (the path between components that a message follows) with the errors that occur within a system. Such correlation can be used to detect the root cause of an error in the system. [8] makes use of *Decision Trees* in order to detect failures in a shopping site.

*Principal Component Analysis* (PCA) has been utilized in [9] to classify the frequency of component interactions in internet services into normal and anomalous behavior. Another significant work [10] makes use of variable length *n-grams* to model call and return order between components in distributed systems. In our opinion, such a method is not suitable for the service model since in services composition, concurrent services are called and return data at the same time (e.g. multiple services providing different outputs to the same service at the same time).

Other related error detection techniques utilize logs of the visited internet pages [11], CPU-instructions and function-calls [12], and messaging between components [13]. In sensor networks, [14] examines simple metrics on network performance.

## VI. Conclusion

In this paper, we have presented our work on the utilization of PCFG modeling for service composition in sensor networks. We have discussed and demonstrated (via simulations) the advantages of our scheme over previous service composition efforts. We believe that other domains, such as error detection in software, and policy-based service composition, can also benefit from this approach. We leave these directions as future work, along with a PCFG modeling where the rules are labeled with performance metrics. Moreover, we plan to do a more detailed evaluation of our methodology, especially in network-based metrics such as communication overhead.

## References

[1] Geyik, S. C., Szymanski, B. K., Zerfos, P., Verma, D., *Dynamic Composition of Services in Sensor Networks*, IEEE SCC 2010.
[2] Geyik, S. C., Szymanski, B. K., *Event Recognition in Sensor Networks by Means of Grammatical Inference*, IEEE INFOCOM 2009.
[3] Wright, J., et al., *A Model-Driven Approach to the Construction, Composition and Analysis of Services on Sensor Networks*, ACITA 2010.
[4] Kiciman, E., Fox, A., *Detecting Application-Level Failures in Component-Based Internet Services*, IEEE Trans. on Neural Networks, Vol. 16, Iss. 5, Sept. 2005, Page(s): 1027-1041.
[5] Silva, L. M., *Comparing Error Detection Techniques for Web Applications: An Experimental Study*, IEEE NCA 2008.
[6] Salfner, F., Lenk, M., Malek, M., *A Survey of Online Failure Prediction Methods*, ACM Computing Surveys Journal, Vol. 42, Iss. 3, March 2010.
[7] Zhang, Z., et al., *Gingko: Correlating Causal Paths in Distributed Systems*, IFIP Int. Conf. on Network and Parallel Computing - Workshops, 2007.
[8] Chen, M., Zheng, A., Lloyd, J., Jordan, M., Brewer, E., *Failure Diagnosis using Decision Trees*, ICAC 2004.
[9] Wu, L., Cheng, L., Qiu, X., Qiao, Y., *A Statistical Approach to Detect Application-Level Failures in Internet Services*, IEEE FSKD 2009.
[10] Jiang, G., Chen, H., Ungureanu, C., Yoshihira, K., *Multiresolution Abnormal Trace Detection Using Varied-Length n-Grams and Automata*, IEEE Trans. on Systems, Man, and Cybernetics Part C: Applications and Reviews, Vol. 37, No. 1, Jan. 2007.
[11] Bodik, P., et al., *Combining Visualization and Statistical Analysis to Improve Operator Confidence and Efficiency for Failure Detection and Localization*, ICAC 2005.
[12] Kasick, M. P., Gandhi, R., Narasimhan, P., *Behavior-Based Problem Localization for Parallel File Systems*, HotDep 2010.
[13] Hirota, E., et al., *Multilayer Failure Detection Method for Network Services Based on Distributed Components*, IEEE/IFIP NOMS 2010.
[14] Ramanathan, N., Chang, K., Kapur, R., Girod, L., Kohler, E., Estrin, D., *Sympathy for the Sensor Network Debugger*, SenSys 2005.