Research Article

Web Services Integration Strategy with Minimum Hops Constraint in Wireless Sensor Networks

Wen Ouyang and Min-Lang Chen

Department of Computer Science and Information Engineering, Chung Hua University, 707 Sec. 2, WuFu Road, Hsinchu 300, Taiwan

Correspondence should be addressed to Wen Ouyang; ouyang@chu.edu.tw

Received 28 June 2013; Revised 30 October 2013; Accepted 31 October 2013; Published 6 January 2014

Academic Editor: Chang Wu Yu

Copyright © 2014 W. Ouyang and M.-L. Chen. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Both wireless sensor networks and service-oriented architectures (SOA) are very popular distributed computing paradigm. Web service is a widely accepted implementation of SOA. However, web service is based on Internet protocols and requires adaptation to suit the restrictions posed by WSNs. The development of Internet of Things has triggered research of web services of WSNs which made the consolidation of these technologies possible. At the same time, web service integration enables the support of more functionalities since many applications rely on not just one web service, but a whole school of them. Thus, how to compose and integrate different web services efficiently to provide complicated services becomes an essential topic. This paper investigates a problem which minimizes the number of hops of web services while integrating these web services to finish a set of tasks. We call this problem Minimum Hops of Service Integration Problem. It is proved that, when there are no precedence relationships between the tasks, the decision problem is NP-complete. This implies that this web services integration problem is NP-hard. For the case when the relationships between the tasks are in linear order, a polynomial-time, optimal web service integration algorithm, using greedy strategy, is provided.

1. Introduction

Wireless sensor networks (WSNs) are used to monitor the concerned events through the collaboration between sensor nodes via wireless communication [1–3]. Recently WSN technologies have been recognized as one of the most important technologies that will seriously impact the world. Due to the advancement of the manufacturing technologies, especially on the communication and battery productions, tiny size of sensors can be equipped with the capabilities of sensing, wireless communication, and data processing. These types of sensors can not only sense and detect targets and changes in their environments but also process the collected data and transmit the data back to the data collection center or base station via wireless communication. The users can thus retrieve the status of the environments and develop interesting application accordingly. WSNs are to deploy massive amount of sensors in the sensing region to collect all kinds of environmental information and then pass the information via wireless network to the base station (sink) and then to the back-end user or manager via the Internet or satellites.

WSN has gained much popularity because of the advanced development of microelectromechanical systems (MEMS) and its wide range of monitoring/tracking applications. Applications of WSNs range in diversified areas [3] such as military, environment, health, home, and other business fields. For military applications, sensors can be used for people identification, battlefield monitoring, enemy detection, and so on. Our living environment can be monitored by sensors to detect events such as forest fire, flooding, pollutions, and volcano eruption. Sensors can also be applied to monitor the health conditions and behaviors of people. Intelligent home also relies on sensors to let people monitor and control the devices in our home. Other applications such as manufacturing automation requires robots equipped with sensors to work together to complete sophisticated work. Traffic control and automobile tracking are also popular applications.
Since the sensors may be deployed randomly to the sensing areas, they need to be able to self-organize using specified protocols to form a communication network so that all sensors in the same network can communicate with each other and transmit the collected data back to the base station. Due to the reason that the required amount of deployed sensors is huge, the sensors have to satisfy the requirements of low cost, low energy consumption, and easy deployment. At the same time, they are programmable, equipped with environment-sensing devices, and can be dynamically reorganized.

Several factors can influence the WSN design [1, 2]. One of them is fault tolerance which means the abilities to sustain the WSN functionalities without any interruptions no matter what type of node failure occurs. Another one is scalability which means the sensor density endurance in the WSNs. Developed schemes for WSNs should work no matter how sparse or how dense the network is. The other factors are hardware constraints, sensor network topology, deployment environment, transmission media, and power consumption. Power consumption is actually a big concern for WSNs due to the reachability of these sensors after deployment. Other factors [3] such as the application's design objectives, cost, and system constraints should also be considered.

Due to the advancement of Internet of Things, more complicated WSNs [4] integrated with heterogeneous devices and multidiscipline systems are a trend to be. These systems have to satisfy many performance requirements such as scalability and security. At the same time, software engineering concerns such as requirement evolvement due to product improvement make the adaptability an important factor to consider. There are many software architecture models for WSNs. The architecture model has to be able to satisfy the rigorous requirements for the WSNs. One popular model is to incorporate a software middle layer to the system [5]. To increase the flexibility, service-oriented middleware is proposed and many related issues are discussed [5–11].

Service-oriented computing paradigm which focuses on loose-coupling software architectures has been much investigated due to its highly modularized properties and platform independent features which enhance system flexibility and adaptability. It relies on service-oriented architecture (SOA) concepts to create large-scale software applications whereas, most of the time, a large set of interacting web service-based components are involved. Nowadays, the challenges for information technology departments have moved from internal processes management to efficiently collaborating with new ideas, functionalities, and partners so that the corporation can be competitive enough [12, 13]. The flexibility of SOA makes it right there for this purpose. However, the SOA practices face many challenges and obstacles although many enterprises, such as Intel, Microsoft [14], Oracle [15], IBM [16], and others [17–19], have endorsed and/or adopted the SOA philosophy in their products or internal business processes.

Web services are distributed web applications which identify a popular implementation of SOA with standard models and protocols. This technology involves open standards such as XML, SOAP, and HTTP. The mechanism involves service providers, service requestors, and service brokers. Although both WSN and web services are taking advantages of distributed computing paradigm, their technologies are far different from each other. Web services are based on Internet protocols. WSNs are based on MEMS with all kinds of resource constraints such as energy, computing power, and its very own set of communication protocols and architectures. The development of Internet of Things has brought these two together [20] with a lot of mending to do.

It is obvious that SOA inherits the concept of component-based software development and thus the integration and composition of SOA play an essential role in the development process. Natis [21] and McCormick [22] point out six keys to the success of SOA which include to invest integration infrastructure and to design service systematically. Another one is to anticipate obstacles which foresees the challenges in it. This reveals how challenging it is to make SOA practice a success, and it also indicates that component integration requires quite much attention and plays an essential role when applying SOA into practices. The performance of the software application heavily relies on the way the services are composed or integrated to support the corresponding applications.

Many of the current SOA developments are still focused on the static software application. However, the issues raised when software applications are composed or integrated dynamically have gained more and more popularity. Dynamic business process modeling has been viewed as a way to improve the organization competition. This can be applied to the following three dynamic software application situations.

1) When for the first time an application is composed, it follows that maybe many web services are out there in the network, waiting to be employed. The choices made in picking up appropriate web service will affect the efficiency of the application greatly.

2) When fault tolerance strategy is required for critical systems, a back-up plan is always needed. There are always times when a node in the network fails to perform its functions. Some web service providers may want to provide redundant node to avoid out-of-service situations. This is necessary in certain critical situations and systems depending on how critical the functionality is and how nonreplaceable the web service is. In certain situations, when the web services are replaceable, the problem of how we can make a good replacement for this web service is also a topic worth discussion.

3) When the software allows dynamic composition, for example, dynamic business processes or some game software which may allow game players to develop their own games dynamically with a set of ready-to-use software modules, the software development process always requires software development professionals to engage in the implementation work. The users are contributors of ideas and suggestions in the normal paradigm of software development processes.
More advanced platform of collaborative development allows the users to take part in the development by themselves. The trend can be traced back to many open online services like Google Video and YouTube till Wikipedia which encourages the users to also be the authors to increase the interactions between all the users. The exact same concept can also be applied in the software development. With web services or SOA in mind, the time for this new paradigm to be realized may be near. For example, for game players, they can reuse the existing building blocks to create new games by themselves and even building new game components using existing services available. In this case, when there are many supporting web services, how to choose the existing components more wisely is essential to the performance of the resulting software.

Lin [23] proposed the idea of accountability of web services. Web services own the advantages of platform independence, component reusability, and flexibility. However, this loosely coupled structure also poses potential problem of the quality of service since there is always a concern about the quality of the whole system while openly integrating all the services from different resources and platforms via network. Thus, an in-depth mechanism is required for making sure the service quality will not suffer. The idea is to establish a service accountability supervision scheme to make sure the services, especially from external resources, can maintain performance and stability. Managing business integrity has been considered [24], by the IBM Research Global Technology Outlook, as one of the systems development trends that will significantly impact the information technology in the near future. This involves the management of policy integrity, process integrity, and core entities information integrity throughout the whole enterprise.

Many studies have been focused on service-oriented architecture [13, 25], especially on applying web services. Quite a few of them are discussing how to compose these web services or how to apply these web services. Some studies are investigating the web services’ nonfunctional properties, for example, how to make web services secure, how to provide highly reliable web services, and how to evaluate web services’ performance.

Following the trend of complicated application requirements and the fast advancement of Internet technologies, the issue with applying web services is not just how to find a single web service which can accomplish one single function. It is essential that we need to satisfy the need of applications with complex requirements. Thus, how to dynamically integrate different web services to satisfy the changing world and achieve better performance is an important problem to solve.

Web service composition/integration has been a hot topic. However, the focus has been on the composition models, frameworks, and mechanisms [26] such as how to extend the WSDL language and how to build the QoS model.

This work investigates the web service integration issues, which consider how to choose a set of web services to satisfy the functional requirements of a pool of tasks and also reduce the web service response time. The idea is based on the concern that the communication time used when hopping between web services contributes the most of response time for the most of the time. To reduce the hop count between web services usually means that we can effectively reduce the response time and the network traffic.

We propose a novel problem of web service integration, that is, how to integrate the existing web services in order to satisfy a pool of tasks and, at the same time, to minimize the web service hop count with the preliminary work published in [27]. This problem plays important roles when the response time is an important factor in evaluating the performance of the applications. This paper considers two types of task precedence relationships. The first case concerns that there are no relationships between the tasks. The second one assumes that the tasks are required to be executed in a specified linear order. We also provide a polynomial-time, optimal greedy solution to integrate the web services in order to achieve the goal of minimizing the hop count of web services for the second case.

The rest of the paper is organized as follows. Section 2 of this paper discusses related work of WSNs, service-oriented WSNs, and web services. Section 3 describes the web service integration problem and the proposed integration system architecture. Section 4 illustrates and analyses the two special cases of the problem and we propose a polynomial-time, optimal algorithm for the second case. Detailed data structure demonstrations for an example are provided in Section 5. Section 6 concludes this work.

2. Related Work

WSNs are used to monitor concerned events through the collaboration between sensor nodes via wireless communication. They have been widely investigated in many issues since it was adopted. WSNs can be classified as terrestrial, underground, underwater, multimedia, and mobile WSN [3]. Terrrestrial WSNs are the typical sensor networks which have been discussed widely. It is usually composed of enormous amount, may be thousands, of inexpensive wireless sensor nodes deployed in a specified area, such as battlefield. The power is limited and thus energy saving is an important issue. Besides, communication protocol is an essential part to consider including data link layer, network layer, and transport layer protocols. Another interesting part is the services which can be provided by WSNs such as localization, coverage, and security. WSNs are usually more vulnerable to attacks and faulty situations possibilities which makes fault-tolerant mechanisms essential when designing and developing WSN-related systems. These mechanisms can be classified as fault-tolerant communications and fault-tolerant data sensing. The purpose is to safeguard the true sensed data to the back end for further processing.

An event of interest can be a single target moving across the sensing area or it may be an event which covers a substantial amount of regions in the sensing area. The job of detecting the former one becomes localization/tracking...
of targets and that of the latter is recognized as the event boundary detections. Mobility is an interesting topic to study in WSN systems. A target can be mobile and it usually is. Sensor nodes can be either static or mobile. Events can also be static or mobile (more likely). Mobile WSNs can take the advantages of mobile nodes to fix problems such as connectivity or coverage issues so that the network can be strongly connected or the whole sensing area can be covered to the maximum degree. Mobile anchor nodes which are aware of their own locations can help in the localization for the other normal nodes too. On the other hand, mobile WSN shows problem in routing since the existing links may be broken when the nodes are moving to different locations. Dynamic routing strategies are required to solve this type of issues. For mobile events or dynamically changing events, the detection work becomes more complicated.

The adoption of WSNs has become a complicated task, especially after the Internet of Things has been seriously studied. Much effort was spent in integrating WSN into Internet of Things in the areas such as IP-based WSNs [28], signaling [29], path and coverage problem [30], and security [31]. Due to their wide application fields, WSNs may need to face constant change of applications and rapid advancement of technologies. To comprehend more complicated situations in a flexible and scalable way, middleware for WSN becomes a solution which can be virtual machine based, database, application driven, message oriented [32], and service oriented [6, 7]. The service-oriented middleware layer approach of WSNs is investigated due to the highly abstract mechanism including interface, its modularity of the architecture which are flexible and scalable, and its standard model and protocols. This approach focuses on building WSNs as services to the applications. Delicato et al. [8] proposed a sensor network architecture based on services in which the sink is a service provider to the applications and a service requestor to the sensor nodes whereas a WSDL document is used to describe the services provided by the sink and sensor nodes. TinySOA [9] was proposed as a service-oriented architecture for WSN. Each WSN provides services which can be internal to the WSN itself or external to outside applications. The architecture components are node, gateway, registry, and server. Nodes are basically the sensor nodes with internal service-providing capability. A gateway bridges the WSN with the external application and knows the services for each node and the whole network. The whole infrastructure is stored in the registry which relies on an internal database to manage the whole system. The last component in TinySOA is the server which is a provider of web services whereas each WSN is considered as a separate web service. Another service-oriented middleware framework for programming [10], called USEME, was also provided so that applications can be easily built using the defined API. USEME provides high-level programming model and languages so that much work will be handled by the middleware such as service publication and discovery, event invocation and communication, and group management. Other service-oriented middleware research [5, 11] works in similar way. Since web service is the most common implementation of SOA, this work adopts the terminology of web service to represent SOA.

From the web service point of view, due to the network technology advancement, network bandwidth and speed have grown greatly. Web service technologies have been widely researched, applied, and integrated including how to efficiently manage the web services. Guan et al. [33] proposed a web services management structure named FASWSM trying to take care of this issue. Web services can be plugged into the application server as a web service adapter to reduce the web services management efforts.

Some researches explore the method of combining different web services to satisfy complicated function requests. Cheng et al. [34] proposed an approach to integrate existing web services and implement a web service integration tool for users who are not familiar with programming skills. Users can compose new services by combining the existing service modules. Mohanty et al. [35] used finite state machines to modularize different web services in order to integrate them to new and more complex network services.

Since web service architecture poses open structures to the outside world, security problems are introduced. Many studies investigate them. Tang et al. [36] presented a comprehensive performance evaluation to understand the security issue of web services. Siddharth and Liu [37] presented a framework for enhancing web services security to prevent DoS or DDoS attacks when designing and implementing web services. Gutiérrez et al. [38] proposed a web services security mechanism as a process that adds a set of specific stages into the traditional phases of WS-based systems development to provide more security. Others like Wu and Weaver [39] proposed, for federated trust management, a framework to facilitate security token exchange across autonomous security domains. Li et al. [40] presented a policy language for adaptive web services security framework and proposed a mixing reasoning framework based on rule and ontology.

Besides security concern, other web services' nonfunctional properties have also drawn attentions. Abramowicz et al. [41] discussed the web services' reliability from the perspectives of both providers and clients. Zo et al. [42] provided a basis to measure reliability of an application system that is integrated using web services. Chan et al. [43] surveyed and addressed applicability of replication and design diversity techniques for reliable web services and proposed a hybrid approach to improving the web services availability.

However, in some applications, the nonfunctional concerns for composing or integrating web services move away from just reliability [42]. Menascé [44] has discussed the major QoS issues of web services to be availability, security, response time, and throughput. Many other issues [26] come into the picture such as how to expand the existing web service protocols, standards, or architectures to accommodate the service composition need. Ardagna and Pernici [45] proposed a QoS modeling scheme for web service selection problem by formulating into a linear programming considering the quality matrix of execution time, availability, price, reputation, and data quality. Zeng et al. [46] also provide a quality model for web service composition with integer programming. Lopes et al. [47] present a service
selection middleware based on service quality metadata and user preference.

In our work, we find a new perspective to investigate the web service integration. In the situation of completing a job with many tasks for an application, each of those tasks can be supported by more than one web service. The issue is when the response time plays an important role; we might want to find a set of web services which can not only finish all the tasks, but also finish them as quickly as possible. This is essential when the outside application needs real-time response from the WSN to react promptly enough to protect life and property. Since web services are connected over the network, communication speed contributes most to the response time in most scenarios. Thus, the first step to minimize the response time is to minimize the web service hop count while executing the tasks. The idea is that if there are more than one consecutive task which can be executed within the same web service, the communication time between web services is saved.

### 3. Problem Formulation and an Architecture

This work concerns the problem of web service integration for WSNs; jobs (applications or services) consist of many tasks which are assigned to different web services over the network for execution. The web services can be WSN related or nonrelated. WSNs can provide both internal and external web services [9]. Since nowadays the amount of web services is increasing tremendously, it is very likely that a task can be supported by several web services. Our work deals with the situations which satisfy the following conditions.

1. All web services are located on different WSNs or machines and require network connections to communicate with each other; otherwise, we can consider them as the same one web service with more than one functionality.

2. The execution time for tasks is negligible compared to network communication time.

3. Each web service is capable of relaying the task requests and associated data to the next web service if necessary. This can further speed up the application finish time since each web service can propagate its results along with other required information to the next web service, instead of passing the result immediately back to the application and letting the application handle the next task assignment. However, even in the situation that the application requires those intermediate results back right away, to minimize the web service hop count should still facilitate the shortening of response time.

The architecture proposed for this web service integration problem is shown in Figure 1. On the Internet side, there are many applications for different types of devices, web services, and directory services. On the other side are WSNs which are used to monitor and track the fields. Each WSN is associated with a gateway which communicates with the outside. The gateway also serves as an internal service directory since each node in the WSN can also be an internal service provider. Between the WSN world and Internet world is the Web Service Integration System (WSIS) which has several components: a WS Integration Agent, a WS Search Agent, and a WS Database. A service requestor—maybe an application or service from a terminal device—sends a job integration request, containing associated tasks information, to the WS Integration Agent which in turn sends request to the WS Search Agent. The WS Search Agent collects information of supporting web services from WS Database and passes related information back to the WS Integration Agent which then decides the way in which the tasks are supported by the web services according to the appropriate methods. Then, the WS Integration Agent invokes those corresponding web services and collects results. Due to the dynamic web service publishing environment over the network, this service integration request can be executed periodically, or while a service requestor either is first built or encounters a task change depending on the requirement of the service requestor. The web service integration agent can apply different strategies, depending on the characteristics of the tasks precedence relationships, for web service integration. The job information can be stored in the WS Database for later use. After the first invoking, the WS Integration Agent can communicate directly with the web services and then get the result back.

The WS Search Agent works by collecting matching web services information from the WS Database or going to the network, most of the time using UDDI or other directory services, to look for more complete list of web services. The search work can be specified by the service requestor using different modes depending on performance concerns. For example, if the real-time response is a concern for the service requestor, the specified mode can be “static” and the search agent will only search the WS Database for matching web services. And if the specified mode is “dynamic,” the search agent will go to the Internet jungle for more matching web services information, pass them to the WS Integration Agent, and store them in the web services database. The integration system can be deployed in a sense that there are multiple WS Search Agents and multiple WS Integration Agents for load sharing purpose to enhance the real-time performance of the system. Or, a master-slave WS Search Agent subsystem can also replace a single search agent for load balancing. The same concern applies to WS Integration Agent too. Furthermore, in case that a WSSA is idle, it can do update work as a background job to enrich the collections in WS Database.

A service requestor can interact with the WS Integration Agent in several different ways. The first one is that it has fixed requirements of functionalities and it sends service requests to the agent. After that, it saves the information in its own storage and uses those pieces of information in the storage since then. It may go to WS Integration Agent again for help only when either there are modifications of the application or service or the chosen web services encounter some problems and cannot function properly. Of course, it can also send information to WS Integration Agent every time it needs services which may cause serious performance issues when
real time is a concern. Another proper time for web service information update is when the service requester is idle.

While assigning tasks to those supporting web services, there may be many combinations of solutions. The hop count of a task assignment is defined as, by applying that specific task assignment, the number of links that an application uses to transport information between web services. We call this problem Minimum Hops of Service Integration Problem. The following example shows two different ways to integrate the services of an emergency management application in Figures 2 and 3. The application requires the tasks of (a) emergency detection service, (b) emergency unit dispatch service, and (c) notification service to conclude the job. The first case shows that the tasks are carried out by three web services while the second uses only two web services, by choosing a web service which can do both (b) and (c), leading the hop counts to be four and three, respectively. This simple example demonstrates that, by properly choosing web services, the number of web services hops, and thus the response time, can be much improved.

4. Two Special Cases and an Optimal Greedy Algorithm

In this section, we consider two special cases of the Minimum Hops of Service Integration Problem. The first case is proven to be NP-hard. For the second case, we proposed a polynomial time implementable, optimal algorithm to achieve the goal of minimum hop count.

The first case considered is where there exist no precedence relationships between the application tasks. We would like to find the web services assigned to tasks with minimum number of hop counts. This problem, no precedence task assignment (NPTA) problem, is equivalent to the problem of using a minimum number of web services to cover all the tasks. It can be proved to be NP-hard with the minimum set cover problem [48] reduction.

The set cover problem is considering the situation of giving an input of several sets. These sets may have some elements in common. The problem is to find the minimum number of sets so that the sets chosen contain all the elements which are in any of the sets. This problem is NP-hard [48].

**Theorem 1.** The NPTA problem is NP-hard.

**Proof.** We can reduce any set cover problem to one of the NPTA problems by the following steps:

1. converting each element of the set cover problem into a task in the NPTA problem,
2. converting each set of the set cover problem into a web service in the NPTA problem.

A solution which leads to the minimum number of web services to finish the execution of an application in NPTA problem is corresponding to the minimum number of sets to cover all the elements in the set cover problem. Thus, the NPTA problem with minimum hop count constraint is NP-hard.

**Theorem 2.** The web services integration problem with minimum hop count constraint is NP-hard.

**Proof.** This is trivial from Theorem 1 since case 1 is a subset of the web service integration problem.

The test cover problem [49] is also equivalent to the first case of task assignment problem. Many heuristic algorithms [49–54] have been proposed for the minimum set cover or test cover problem, and a recent heuristic work in test suite minimization problem using concept analysis [55] can also be applied directly to solve this problem.
The second case we are focusing on solving is what can be done and how good they are when the tasks have to be finished in a specified linear order, called Linear Order Web Service Integration Problem (LOWSIP). That is, the tasks can be sorted in the order of which they need to be finished. Task $i$ has to be done before task $j$ can be proceed if and only if $i < j$. For any two consecutive tasks called task $i$ and task $i+1$, if these two tasks are placed on different web services, then the web service assigned to execute task $i$ needs to carry over information to the web service that is to execute task $i + 1$ via SOAP protocol. In other words, if two consecutive tasks are assigned to the same web service, then there is no information passing between web services.

We provide an optimal algorithm, OPT, to integrate web services using greedy strategy in the sense of minimizing the hop count between web services in order to speed up the response time. The OPT algorithm is shown in the flow chart as in Figure 4 and works as follows. Recall that there are a set of linear-ordered tasks to be executed and a set of web services each of which is capable of performing certain tasks.

1. Divide each web service $WS_i$ into the minimum number of miniweb services $w_{s_{i1}}, w_{s_{i2}}, \ldots, w_{s_{ij}}$ where each miniweb service can serve a set of maximal consecutive tasks. Each miniweb service contains trio information: its original web service, the starting task ID, and the last task ID.

2. Sort the miniweb services in the increasing order of the first tasks they serve. If two miniweb services share the same starting tasks, then remove the one with the smaller last task ID. Keep the leftover miniweb services in an ordered miniweb service (OMWS) list.

3. Loop Steps 4 and 5 through all the tasks starting from task one. Let the current task be task $i$. 

Figure 2: Emergency management example using three web services.

Figure 3: Emergency management example using two web services.
(4) Task \( i \) is assigned to the first miniweb service, \( w \), remaining in the OMWS list.

(5) Update the OMWS list to remove the redundant miniweb services and the just assigned one. (a) For all miniweb services remaining in the list, update their start task ID to remove the tasks already covered by \( w \). After this, if a miniweb service \( a \) is covered by another one \( b \), remove \( a \) from the OMWS list. (b) Remove \( w \) from the list.

(6) The true web service assigned to each task can be retrieved from the first part of the miniweb service trio information.

An example is used to demonstrate the concepts and relationships between the tasks and web services of OPT algorithm. In Figure 5, the original task and web service correlations are shown. It shows, for instance, that web service \( W_1 \) can serve tasks \( T_1 \), \( T_3 \), and \( T_4 \).

Figure 6 shows the result of the web services’ breaking up into miniweb services (Step 1). For example, web service \( W_1 \) is breaking up into miniweb services \( w_{11} \) and \( w_{12} \), where \( w_{11} \) serves \( T_1 \) and \( w_{12} \) serves \( T_3 \) and \( T_4 \).

Figure 7 (Step 2) exhibits the relationships after the sorting of all miniweb services. The miniweb services are lined up in the order of \( w_{11} \), \( w_{12} \), \( w_{21} \), \( w_{22} \), \( w_{31} \), \( w_{32} \), \( w_{41} \), \( w_{42} \), \( w_{51} \), \( w_{52} \), and \( w_{53} \). Miniweb service \( w_{31} \) is placed before \( w_{12} \) since it has the starting task ID, \( T_2 \), before that of \( w_{12} \), \( T_3 \).

Then, the task assignment and redundant mini task removing start. From Step 4, task \( T_1 \) is assigned to \( w_{11} \) which is actually \( W_1 \). There are no redundant miniweb services at this time (Step 5). The process continues. Task \( T_2 \) is in \( W_2 \). Task \( T_3 \) is in \( W_1 \) which makes \( T_4 \) in \( W_1 \) too. Now, \( w_{31} \) has \( T_5 \), \( w_{32} \) has \( T_5 \), and \( w_{31} \) has \( T_6 \), \( T_7 \), and \( T_7 \). Thus, \( w_{31} \) and \( w_{32} \) are eliminated since they are redundant. The result after looping Steps 4 and 5 is in Figure 8. The dash lines in Figure 8 are the removed link and eliminated miniweb services due to the duplication of the coverage of tasks (Step 5). The final result can also be viewed from Figure 8 which reveals the web service serving each task. Thus, to complete all tasks \( T_1 \) to \( T_{10} \), the web service sequence used for the tasks is \( W_{11}, W_{21}, W_{31}, W_{41}, W_{51}, W_{61}, W_{7}, W_{8}, W_{9}, \) and \( W_{10} \). The number of web service hops for this example is five. The result is an optimal solution which means any solution other than this one would have at least five web service hops.

**Theorem 3.** Algorithm OPT is optimal.

**Proof.** The algorithm OPT can be proved to be optimal. Assume that the solution generated from OPT algorithm, \( S_{opt} \), is not optimal for a tasks-web services scenario. Then, there exists a real optimal solution, say \( S_r \), which leads to a minimum number of web service hops while \( S_{opt} \) does not. That is, this real optimal solution can divide the tasks into fewer chunks than \( S_{opt} \) can, where each chunk of tasks is a set of consecutive tasks which are served by the same web service. Let is compare the first chunks in both \( S_{opt} \) and \( S_r \). Since OPT algorithm always picks up a web service which supports the current task and supports the most other remaining tasks, the size of the first chunk in \( S_{opt} \) must be larger than or equal to the size of that in \( S_r \). So, we can replace...
the first web service in $S_r$ with the first web service in $S_{opt}$ without affecting the number of web service hops for $S_r$. This process continues and we find out that $S_r$ can be completely converted into $S_{opt}$ without increasing the number of web service changes, so there is no way for $S_r$ to have a less number of web service changes than $S_{opt}$. Thus, we prove that OPT produces an optimal solution to have the least number of web service changes. This algorithm requires polynomial time to complete the task assignment work.

5. Detailed Data Structure Illustration

We use another example to illustrate the details when the data structures used between web services and tasks are a Boolean matrix. In Table 1, there are 20 tasks and 10 web services, and the matrix exhibits the supporting relationships between them. Each $W_i$ represents a web service, and the Boolean values following $W_i$ are the supporting conditions between this web service and all tasks. The value 1 means that $W_i$ supports the corresponding task whereas value 0 otherwise. As in the table, web service $W_i$ supports tasks $T_{10}, T_{17}, T_{18}, T_{19}$ and is with $S_{opt}$. The web services are divided up into miniweb services, each of which supports maximum number of consecutive tasks. The next step is to sort the miniweb services according to their starting task, and the result is in Table 2. After that, the miniweb service with the smaller last task ID is removed, if two miniweb services share the same starting tasks. Also, after each assignment, the OMWS list is updated.

The OPT algorithm then repeatedly selects the best miniweb services and updates the OMWS list until all tasks are satisfied. The final updated OMWS list is shown in Table 3. The web service assignment result is in Figure 5.

Starting from task $T_j$, OPT algorithm picks up the miniweb services in the OMWS list which supports $T_j$ and also supports most other tasks, that is, $ws_{k_1}$. Thus, tasks $T_j \sim T_{k_1}$ will be assigned to $ws_{k_1}$. The next step is to update OMWS list. After that, the algorithm works by starting from task $T_j$, and the miniweb service left which supports $T_j$ and is with the most tasks, $ws_{k_2}$, to support will be picked to execute $T_{k_2}$, and so on.

The number of web service hop counts is six, as shown in Figure 9, and this is one optimal result derived by the OPT algorithm.

---

**Table 1: Web services and task relationship.**

| Task   | Web services supporting relationships |
|--------|---------------------------------------|
| $T_1$  | $W_1 (0,1,1,1,0,0,0,0,1,0,0,1,1,1,1,0)$ |
| $T_2$  | $W_2 (1,1,0,0,0,1,1,0,1,1,1,1,0,0,1,1,0)$ |
| $T_3$  | $W_3 (0,0,1,1,1,0,0,1,1,1,1,1,0,0,1,1,0)$ |
| $T_4$  | $W_4 (1,1,0,0,0,1,0,0,1,1,1,1,1,1,1,0,1)$ |
| $T_5$  | $W_5 (0,1,0,0,1,0,0,1,0,0,1,1,1,1,1,0,0)$ |
| $T_6$  | $W_6 (1,1,0,0,0,0,1,1,1,1,1,1,0,0,0,0,1)$ |
| $T_7$  | $W_7 (0,1,0,0,0,1,0,0,1,0,0,0,1,1,1,1,0)$ |
| $T_8$  | $W_8 (0,1,0,0,0,0,1,1,1,1,1,1,0,0,0,0,0)$ |
| $T_9$  | $W_9 (1,1,0,0,0,1,0,0,1,0,0,1,1,1,0,0,1)$ |

---

**Table 2: Tasks and mini web services relationship after sorting.**

| Task | Mini web services |
|------|------------------|
| $T_1$ | $ws_{11} (1,2)$, $ws_{12} (1,3)$, $ws_{21} (1,2)$, $ws_{22} (1,3)$ |
| $T_2$ | $ws_{11} (2,3)$, $ws_{12} (2,4)$, $ws_{21} (2,4)$ |
| $T_3$ | $ws_{11} (3,5)$, $ws_{41} (3,3)$ |
| $T_4$ | $ws_{62} (4,6)$ |
| $T_5$ | $ws_{22} (5,7)$, $ws_{12} (5,7)$, $ws_{22} (5,6)$, $ws_{31} (5,5)$ |
| $T_6$ | $ws_{51} (6,6)$, $ws_{22} (6,7)$, $ws_{12} (6,6)$ |
| $T_7$ | $ws_{12} (7,8)$ |
| $T_8$ | $ws_{31} (8,10)$, $ws_{81} (8,10)$, $ws_{91} (8,13)$, $ws_{81} (8,10)$, $ws_{81} (8,9)$ |
| $T_9$ | $ws_{23} (9,11)$, $ws_{73} (9,10)$ |
| $T_{10}$ | $ws_{5,3} (10,10)$, $ws_{81,2} (10,14)$ |
| $T_{11}$ | $ws_{4,3} (11,12)$ |
| $T_{12}$ | $ws_{14} (12,14)$, $ws_{5,4} (12,13)$ |
| $T_{13}$ | $ws_{8,4} (13,15)$ |
| $T_{14}$ | $ws_{2,4} (14,14)$, $ws_{8,4} (14,16)$ |
| $T_{15}$ | $ws_{14} (15,19)$, $ws_{8,4} (15,16)$, $ws_{5,4} (15,18)$, $ws_{5,2} (15,17)$ |
| $T_{16}$ | $ws_{5,5} (17,18)$, $ws_{5,3} (17,19)$ |
| $T_{17}$ | $ws_{4,4} (18,20)$ |
| $T_{18}$ | $ws_{6,4} (19,19)$ |
| $T_{19}$ | $ws_{4,5} (19,19)$, $ws_{8,6} (19,19)$, $ws_{8,5} (19,20)$, $ws_{5,5} (19,20)$ |
| $T_{20}$ | $ws_{8,3} (20,20)$ |
6. Conclusion and Future Work

WSNs have attracted much attention and have been integrated into bigger infrastructures such as Internet of Things. Service-oriented middle layer architecture of WSN enables the adaptability and flexibility of the whole systems. In this work, we propose a problem to enhance the WSN web service integration to reduce the hop count of web service integration with the reduction of traffic and response time in mind. The problem is called Minimum Hops of Service Integration. Two special cases are investigated. The first one is for the case when there are no precedence relationships between the tasks and it is proven to be NP-complete. However, when the task precedence relationships are linearly ordered, we develop a polynomial-time greedy algorithm, OPT, to solve this problem.

In the future, we will investigate the effectiveness of different web services integration methods on other non-functional factors in order to develop a more efficient and comprehensive way of integrating web services.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgment

This work is sponsored by NSC of Taiwan under Grant no. NSC 100-2632-E-216-001-MY3-3.

Table 3: Updated miniweb services.

| Start task | Miniweb services | Start task | Miniweb services |
|------------|----------------|------------|----------------|
| T_1        | ws_{41} (1, 3)  | T_{11}     | ws_{4,5} (11, 12) |
| T_2        | ws_{5,1} (2, 4) | T_{12}     | ws_{5,4} (12, 14) |
| T_3        | ws_{1,1} (3, 5) | T_{13}     | ws_{10,4} (13, 15) |
| T_4        | ws_{6,2} (4, 6) | T_{14}     | ws_{4} (14, 16) |
| T_5        | ws_{6,2} (5, 7) | T_{15}     | ws_{6,4} (15, 19) |
| T_6        | ws_{2,2} (6, 7) | T_{16}     |                  |
| T_7        | ws_{3,2} (7, 8) | T_{17}     | ws_{3,5} (17, 19) |
| T_8        | ws_{3,3} (8, 13)| T_{18}     | ws_{5} (18, 20)  |
| T_9        | ws_{5,3} (9, 11)| T_{19}     | ws_{5,5} (19, 20) |
| T_{10}     | ws_{5,2} (10, 14)| T_{20}    | ws_{5,1} (20, 20) |

References

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, “A survey on sensor networks,” IEEE Communications Magazine, vol. 40, no. 8, pp. 102–105, 2002.
[2] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, “Wireless sensor networks: a survey,” Computer Networks, vol. 38, no. 4, pp. 393–422, 2002.
[3] J. Yick, B. Mukherjee, and D. Ghosal, "Wireless sensor network survey," Computer Networks, vol. 52, no. 12, pp. 2292–2330, 2008.
[4] L. Mainetti, L. Patrono, and A. Vilei, "Evolution of Wireless Sensor Networks towards the Internet of Things: a Survey," in Proceedings of the 19th International Conference on Software, Telecommunications and Computer Networks (SoftCOM ‘11), pp. 1–6, September 2011.
[5] K. K. Khedo and R. K. Subramanian, “A service-oriented component-based middleware architecture for Wireless Sensor Networks,” International Journal of Computer Science and Network Security, vol. 9, no. 3, pp. 174–182, 2009.
[6] N. Mohamed and J. Al-Jaroodi, “A survey on service-oriented middleware for wireless sensor networks,” Service Oriented Computing and Applications, vol. 5, no. 2, pp. 71–85, 2011.
[7] N. Mohamed and J. Al-Jaroodi, “Service-oriented middleware approaches for wireless sensor networks,” in Proceedings of the 44th Hawaii International Conference on System Sciences (HICSS-44 ’10), pp. 1–9, January 2011.
[8] E. C. Delicato, P. P. Pires, L. Pirmez, and L. E. da Costa Carmo, “A flexible web service based architecture for wireless sensor networks,” in Proceedings of the 23rd International Conference on Distributed Computing Systems (ICDCSW ’03), p. 730, 2003.
[9] E. Avilés-López and J. A. García-Máciás, “TinySOA: a service-oriented architecture for wireless sensor networks,” Service Oriented Computing and Applications, vol. 3, no. 2, pp. 99–108, 2009.
[10] E. Cañete, J. Chen, M. Díaz, L. Llopis, and B. Rubio, “A service-oriented middleware for wireless sensor and actor networks,” in Proceedings of the 6th International Conference on Information Technology: New Generations (ITNG ’09), pp. 575–580, Las Vegas, Nev, USA, April 2009.
[11] F. Golutowski, J. Blumenthal, H. Matthias, M. Haase, H. Burchardt, and D. Timmermann, “Service-oriented software architecture for sensor networks,” in Proceedings of International Workshop on Mobile Computing (IMC ’03), pp. 93–98, June 2003.
[12] M. P. Singh and M. N. Huhns, Service-Oriented Computing: Semantics, Processes, Agents, John Wiley and Sons, 2005.
[13] A. Maurizio, J. Sager, P. Jones, G. Corbitt, and L. Girolami, “Service oriented architecture: challenges for business and academia,” in Proceedings of the 41st Annual Hawaii International Conference on System Sciences (HICSS ’08), pp. 315–322, January 2008.
[14] MSDN Architecture Center, “Service Oriented Architecture,” http://msdn.microsoft.com/en-us/architecture/aa948857.aspx.
[15] Oracle, "Oracle Service-Oriented Architecture," http://www.oracle.com/technologies/soa/index.html.
[16] IBM, “Service Oriented Architecture-SOA,” http://www-306.ibm.com/software/solutions/soa/.
[17] Sun, “Service-Oriented Architecture (SOA),” http://www.sun.com/products/soa/index.jsp.
[18] M. Nicolai Josuttis, SOA in Practice: The Art of Distributed System Design, O’Reilly, 2007.
[19] Thomas Erl, SOA Principles of Service Design, Prentice Hall, 2007.
[20] G. Moritz, F. Golatowski, C. Lerche, and D. Timmermann, “Beyond 6LoWPAN: web services in WSNs,” IEEE Transactions on Industrial Informatics, vol. 9, no. 4, pp. 1795–1805, 2013.
[21] Y. V. Natis, “Applied SOA: transforming fundamental principles into best practices,” Gartner Research Note Id G00147098, 2007.
[22] J. McCormick, “6 Keys to SOA Success,” Baseline, 2007, http://www.baselinemag.com/article2/0,1540,2139539,00.asp.
[23] K.-J. Lin, “The design of an accountability framework for service engineering,” in Proceedings of the 41st Annual Hawaii International Conference on System Sciences (HICSS’08), pp. 108–116, January 2008.
[24] C.-S. Li, Managing Business Integrity Across Policies, Processes and Information, IBM Information On Demand, 2007.
[25] D. Woods and T. Mattern, Enterprise SOA: Designing IT for Business Innovation, O’Reilly, 2006.
[26] S. Dusdhar and W. Schreiner, “A survey on web services composition,” International Journal of Web and Grid Services, vol. 1, no. 1, pp. 1–30, 2005.
[27] W. Ouyang and M.-L. Chen, “An Optimal web services Integration using Greedy Strategy,” in IEEE Asia-Pacific Services Computing Conference, pp. 568–573, 2008.
[28] S. Hong, D. Kim, M. Ha et al., “SNAIL: an IP-based wireless sensor network approach to the Internet of things,” IEEE Wireless Communications, vol. 17, no. 6, pp. 34–42, 2010.
[29] S. Li, L. Da Xu, and X. Wang, “Compressed sensing signal and data acquisition in wireless sensors networks and internet of things,” IEEE Transactions on Industrial Informatics, vol. 9, no. 4, pp. 2177–2186, 2013.
[30] L. Liu, X. Zhang, and H. Ma, “Percolation theory-based exposure-path prevention for wireless sensor networks coverage in internet of things,” IEEE Sensors Journal, vol. 13, no. 10, pp. 3625–3636, 2013.
[31] F. Li and P. Xiong, “Practical secure communication for integrating wireless sensor networks into the internet of things,” IEEE Sensors Journal, vol. 13, no. 10, pp. 3677–3684, 2013.
[32] S. Hadim and N. Mohamed, “Middleware: middleware challenges and approaches for wireless sensor networks,” IEEE Distributed Systems Online, vol. 7, no. 3, pp. 1–23, 2006.
[33] H. Guan, B. Jin, J. Wei, W. Xu, and N. Chen, “A framework for application server based web services management,” in Proceedings of the 12th Asia-Pacific Software Engineering Conference (APSEC ’05), pp. 1–8, December 2005.
[34] F.-C. Cheng, T.-C. Hung, Y.-J. Chiou, and T.-C. Chang, “Design and implementation of Web service integration tool,” in IEEE International Workshop on Service-Oriented System Engineering, pp. 91–96, October 2005.
[35] H. Mohanty, J. Mulchandani, D. Chenthati, and R. K. Shyamasundar, “Modeling Web Services with FSM Modules,” in Proceedings of the 1st Asia international conference on Modelling & Simulation, pp. 100–105, March 2007.
[36] K. Tang, S. Chen, D. Levy, J. Zic, and B. Yan, “A performance evaluation of web services security,” in Proceedings of the 10th IEEE International Enterprise Distributed Object Computing Conference (EDOC ’06), pp. 67–74, chin, October 2006.
[37] N. Sidharth and J. Liu, “IAPF: a framework for enhancing web services security,” in Proceedings of the 31st Annual International Computer Software and Applications Conference (COMPSAC ’07), pp. 23–30, July 2007.
[38] C. Gutiérrez, E. Fernández-Medina, and M. Piattini, “PWSec: process for Web services security,” in Proceedings of the IEEE International Conference on Web Services (ICWS ’06), pp. 213–222, September 2006.
[39] Z. Wu and A. C. Weaver, “Using web services to exchange security tokens for federated trust management,” in Proceedings of the IEEE International Conference on Web Services (ICWS ’07), pp. 1176–1178, July 2007.
[40] J.-X. Li, B. Li, L. Li, and T.-S. Che, “A policy language for adaptive Web services security framework,” in Proceedings of the 8th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, pp. 261–266, July 2007.
[41] W. Abramowicz, M. Kaczmarek, and D. Zyskowski, “Duality in Web services reliability,” in International Conference on Internet and Web Applications and Services/Advanced International Conference on Telecommunications, p. 165, February 2006.
[42] H. Zo, D. L. Nazareth, and H. K. Jui, “Measuring reliability of applications composed of web services,” in Proceedings of the 40th Annual Hawaii International Conference on System Sciences (HICSS ’07), p. 278c, January 2007.
[43] P. P. W. Chan, M. R. Lyu, and M. Malek, “Reliable Web services: methodology, experiment and modeling,” in Proceedings of the IEEE International Conference on Web Services (ICWS ’07), pp. 679–686, July 2007.
[44] D. A. Menascé, “QoS issues in web services,” IEEE Internet Computing, vol. 6, no. 6, pp. 72–75, 2002.
[45] D. Ardagna and B. Pernici, “Adaptive service composition in flexible processes,” IEEE Transactions on Software Engineering, vol. 33, no. 6, pp. 369–384, 2007.
[46] L. Zeng, B. Benatallah, A. H. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, “QoS-aware middleware for Web services composition,” IEEE Transactions on Software Engineering, vol. 30, no. 5, pp. 311–327, 2004.
[47] F. Lopes, T. Batista, E. Calvalcante et al., “Dynamic and semantic web services composition for ubiquitous computing,” in Proceedings of the 18th Brazilian symposium on Multimedia and the web, pp. 151–160, 2012.
[48] M. R. Garey and D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman and Co., San Francisco, Calif, USA, 1979.
[52] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, MIT Press, 2nd edition, 2001.

[53] M. J. Harrold, R. Gupta, and M. L. Soffa, "Methodology for controlling the size of a test suite," *ACM Transactions on Software Engineering and Methodology*, vol. 2, no. 3, pp. 270–285, 1993.

[54] M. Marré and A. Bertolino, "Using spanning sets for coverage testing," *IEEE Transactions on Software Engineering*, vol. 29, no. 11, pp. 974–984, 2003.

[55] S. Tallam and N. Gupta, "A concept analysis inspired greedy algorithm for test suite minimization," in *ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering*, pp. 35–42, September 2005, SIGSOFT Software Engineering Notes, vol. 31, no. 1, January 2006.
Submit your manuscripts at http://www.hindawi.com