A Practical Approach to Adaptive Service Composition

Dimuthu U. Gamage, Ryan Rybarczyk, Rajeev R. Raje

Department of Computer and Information Science, Indiana University Purdue University Indianapolis Indianapolis, IN 46202, U. S. A.

{dcundupi, rrybarcz, rraje}@cs.iupui.edu

Abstract - **Most web services today are made up of dynamic entities, and because of these entities the services may need to evolve and adapt to their environment over time. This adaptation, by the web services, can occur at any of the service contract levels. In addition, the requirements of the client may change over time based upon their needs and wants. Because of this, the service composition process needs to also be adaptable and dynamic while taking into account the multi-level agreements present within distributed system. This composition process is required when a client has specified requirements that a single service may not completely satisfy but instead may require many services to meet their given needs. Through the creation of a proxy service these needs can then be composed and presented to the client as if they were working within a single service. Additionally, as both service and client requests change or are altered over time the composer also may need to adjust this proxy service to meet both sets of needs. In this paper, we propose a design of an adaptive service composer that will provide a focus on automatic adaptation of semantic and quality of service (QoS) level contracts in compositions. This proposed implementation will then be empirically validated on the existing Enhanced Distributed Object Tracking System (eDOTS).**

Keywords - **Distributed Systems, Software Services, Service Composition, Adaptation.**

1. INTRODUCTION

More and more distributed systems are currently being assembled out of independently created Web Services and this trend is expected to continue in future. As these distributed systems may need to adapt over a period of time due to various reasons, such as performance requirements, their underlying architecture consisting of a composition of individual services also need to be adaptable in nature. These distributed systems may range from simple to complex, in terms of the features that they provide to the clients. For instance, a service could provide relevant athletic event information, but a separate service may be needed to get the weather forecast for those events. In this situation, the use of service composition would save the client from being required to make additional queries to retrieve the desired information. This composition may be a trivial task, in the case of the previous example, or it could involve a complex workflow. This scenario could be avoided, however, if instead the client only interacted with a single point of contact that acts as a front end of the required distributed system. This could be achieved by using a service composer that acts as an intermediary between the composed distributed system and the client and hides the underlying details of how the information is collected from multiple services. Also, a client's request may be multi-level [1] in nature consisting of syntax, semantics, synchronization, and Quality of Services (QoS)-related features and the service composer may need to provide an interface that satisfies such a request.

As appealing the idea of designing such a service composer is, its design is hardly trivial especially when the services that participate in the composition change over a period of time. Such changes will be reflected in the multi-level contracts of services [1] and may be due to the changes in the execution environment (e.g., addition of new resources) or internal changes associated with the services (e.g., upgrades associated with service algorithms). A service composer that is aware of such changes should ensure that the composed distributed system still satisfies a client's requirements, regardless of the changes in the underlying participating services. For that, the composer should re-evaluate the composed system, and carry out appropriate modifications (i.e., search, discover, add, remove or replace services as needed). We refer to such an adaptive service composer as the *composing agent.*

As a concrete application scenario, consider a distributed tracking system that is made up a set of heterogeneous sensor services (such as camera services, and wireless trackers), a filter service, and a fusion service that satisfies certain QoS requirements (such as response time and error). The composing agent can create this tracking system out of existing choices for each type of these services and ensure that the QoS-requirements are met initially. However, over time, the participating camera services may provide lower response time, or less accurate tracking information (due to the environment changes such as network delays, processing and battery limits). In such situations, the composing agent would re-evaluate the system and carry out the necessary changes in the composition (e.g., replace the slow sensor services with more powerful sensor services) to meet the QoS-requirements of the client. However, when such changes happen frequently, the overhead of completely re-evaluating the current composition would not be desirable as it would be an additional burden on the performance of the underlying system (in this case, the distributed tracking system). Therefore, it is important that the composing agent carries out minimal changes to an existing distributed system while still satisfying the client's multi-level requirements. As a result, we are proposing a novel approach to adaptive composition by allowing

the composing agent to either actively or passively request information from both the client and existing Web Services in order to better assist, adapt, and provide the required service composition in the presence of changes.

In this paper, we discuss the following contributions:

1. The implementation of the composition agent for the QoS- and Semantic-level contracts that provides dynamic service composition both in an active as well as a passive manner (this will be referred from here on as eager and lazy composition). This agent serves as a plugin within an existing distributed client application.

2. An empirical validation of the composition agent along with associated performance analysis via a case study consisting of a distributed indoor tracking system (the eDOTS [12-15]).

The rest of the paper is organized as follows: Section 2 compares the proposed adaptive composition approach with existing approaches, Section 3 discusses the design of the adaptive composition framework, Section 4 presents core implementation details of the framework, Section 5 presents the results of experiments on the framework, and Section 6 presents the concluding remarks and the related future research directions.

2. RELATED WORK

In Bracciali et al. [2], the authors present a formal method to adapt components that have mismatching semantics. Their method includes specifying semantics of components, using these component specifications in expressing the adaptor specification, and generating the adapter automatically to enable the communication between components using the prepared component specifications and adapter specifications. This method requires writing adaptor specifications between any two semantically incompatible components to enable the communication between them. Therefore, this approach cannot be used in a dynamic service oriented system where the services are discovered at runtime based on particular user requirements. In contrast, the composing agent proposed in this paper has the ability to enable the communication between set of services and clients dynamically based on the state of the services at that point.

Phatak [17] discusses adaptation techniques of multi-level specifications [1] (i.e., syntax, semantics, synchronization and QoS levels) for dynamically adapting services. That work has shown the importance of specifying different adaptations supported by the services formally in their corresponding specifications. This helps the clients to prepare the adaptation of communication with the services accordingly. We have extended their work to build an adaptive composing agent based on the multi-level specifications of services.

McKinley et al. [10] classify compositional adaptation techniques of middleware. Their studies include design decisions to be made in an adaptive composition and associated challenges. However, their discussions are limited to the use of syntax and semantic compatibility in designing re-composition in adaptive environment. In contrast, our work includes the use of multi-level specifications (especially, the QoS level) in designing the re-composition.

There have been some attempts (such as, Hemer et al. [6], Camara et al. [8]) to present formal approaches for the adaptive composition at the semantic level. However, investigating adaptations of compositions in QoS levels are also important, as many real-time and embedded applications run on strict QoS requirements and these QoS constraints should be maintained even with the changes in the underlying execution environment.

Zhan et al. [3] propose a QoS-aware adaptive composition method that involves policy-driven location-aware service discovery and an adaptation policy to restrict the re-evaluation of the composition. Although the goal of their work is very similar to ours, our methods involve more automation of composition based on heuristics algorithms rather than the manually provided policies to obtain a QoS optimized composed system. Aschoff et al. [16] provide another aspect of QoS-aware adaptive composition by presenting a method of proactively predicting the changes in QoS of the available services and re-evaluating the composition based on the prediction. In contrast, our focus is more on re-evaluating the composition with the changes in the environment, while client can still use the system without being aware of the changes (or with minimum awareness).

3. BACKGROUND AND DESIGN

The task of a service composer, as shown in Figure 1, is to take a request from a client and create a distributed system, out of available services, which meets that client's requirements. This composition process requires that the composer be provided a list of available services and what they offer – the task of gathering this information is termed as service discovery and various architectures have been proposed for this task [4]. The composer will then examine the details related to different levels of specifications associated with each service, select a subset of relevant services, and subsequently, use these selected services to create a distributed system and present a single view of the composed system to the client. We refer to this type of a composer as a *basic composer*.

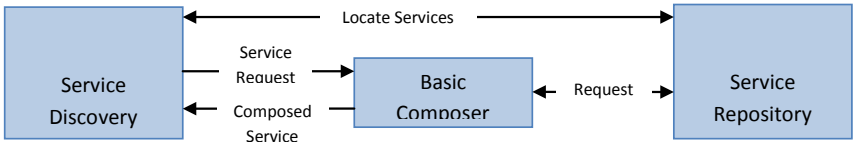

Figure 1 : Basic Composer

The basic composer should provide a generic interface to other entities (e.g., service discovery), so that it can be applied to a particular domain with minimum effort. A service composition framework that includes the basic composer is shown in Figure 2. In that figure, the domain independent and domain dependent pieces are identified separately. Thus, this framework provides plug-in interfaces that allow domain experts to integrate the domain dependent entities with the basic composer with minimal effort. Among the domain independent components provided by the framework, the Repository is a database storage of the service specifications, the Composer provides the necessary algorithms to generate composition specifications from available services, and the Discovery provides interface to clients to query services. As shown in Figure 1, the discovery either presents single service directly from the repository or if a single service is not available to suit the client's requirement, it provides a composed service to the clients. Among the domain specific components Composition Operators provide mechanisms to calculate composed attributes of services, Selection Heuristics provides algorithms that heuristically select services for composition, the Knowledgebase provides the relationships between entities in a particular domain that are used in finding the compatibility of entities in different services, and Service Specifications provides the functional and non-functional attributes of each individual services.

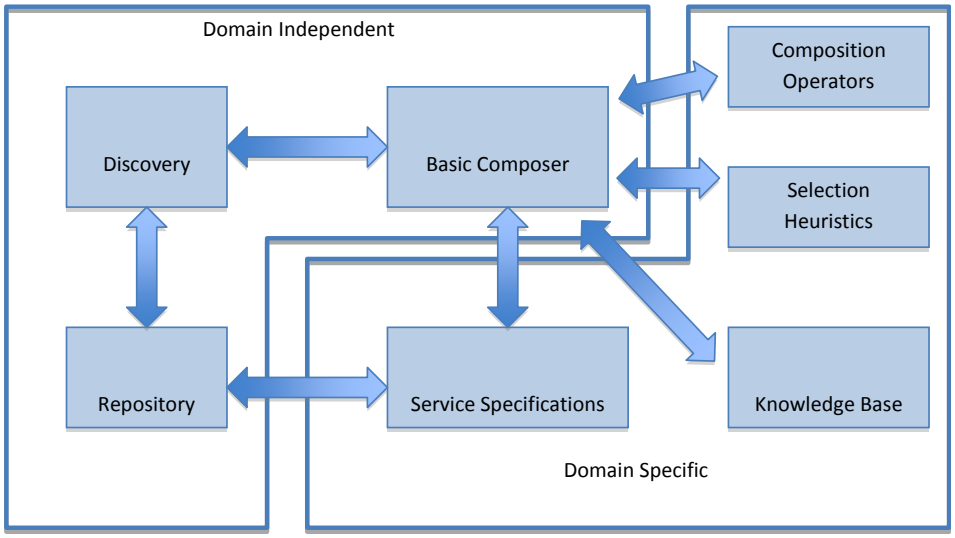

Figure 2 : Architecture of the Basic Service Composition Framework

Although, the basic composer needs to consider all the levels indicated in the specification of services during the composition process, in this paper, we focus solely on the semantic and QoS levels of service contracts [1]. This decision is primarily based upon the ability to easily negotiate these levels between the service and client, as well as the fact that these two levels are the most likely to undergo changes throughout a service's life time. These two layers are described below.

Semantic Level Composition: The composition at the semantic level involves the pre- and post-conditions as specified by the service itself. This level deals with the overall behavior of a particular service within a distributed system. When the client makes a request, the pre- and post-conditions provided must be satisfied and ensured as part of the fulfillment of the query. This provides the user a guide about how the service will behave prior to and after it finishes its required task. This guarantee, provided by the developer of each service that is used to create the distributed system, is utilized by the basic composer in the composition process to meet a client's requirements.

QoS Level Composition: The composition at the QoS level involves utilizing predefined QoS-attributes and their associated values that are guaranteed by the developers of services and the interaction patterns between the services that constitute a distributed system. This composition process will often times require an external intervention and use of heuristics to select a proper set of services from the available ones that satisfy the client's QoS requirements. Heuristics can either be provided by a domain expert or could be based on past execution history.

3.1. Design of an adaptive service composer

The proposed adaptive composer will, in addition to the abovementioned functions of the basic composer, perform the task of observing changes in underlying services, as evident from alterations to their specifications, and to accommodate for these changes while preserving, as much as possible, the same view to the client of the composed distributed system. As indicated earlier, we refer to such an adaptive service composer as a *composing agent.*

The design of the composing agent needs to consider additional attributes such as the adaptive behavior of services, the rigidness of the requirements of clients in an effort to provide a runtime evaluation of service level contracts and associated services in a composition. We are proposing two types of evaluations, namely *lazy* and *eager*, to be incorporated in the design of the composing agent as shown

in the Figure 3. Lazy or passive evaluation takes place in a similar fashion to the basic composer. This lazy evaluation allows the composing agent to query the service repository in a non-invasive manner in order to maintain a directory of available services and their attributes. This method will rely on service instances to provide status updates if any variations in their behaviors take place during the course of their executions. Therefore, there is a certain level of trust (i.e., the entities will behave as expected) that must be defined between the composing agent, the repository, and the client request.

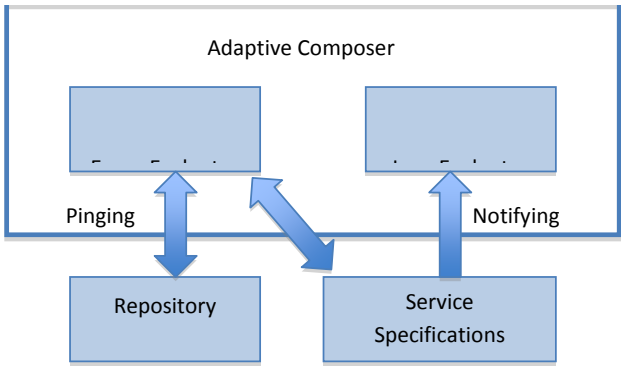

Figure 3 : Architecture of the Adaptive Service Composer or Composing Agent

For eager or active evaluation, we place the responsibility of gathering the most up-to-date information about services on to the composing agent. This will allow the composing agent to query services and the repository in an effort to maintain the most current information about the services. These queries are issued either whenever a service request is made to create the composed system or at an appropriate frequency that is calculated by studying the past history of adaptation of the relevant services. This is especially important in situations where the services are constantly in a state of change. This also will adjust to the needs of the client's request, as it could also be subject to change. Through the eager evaluation process, the composing agent provides a high degree of guarantee of fulfilling the client's requirement as it maintains the most up-to-date information about the services.

The usage of the adaptive service composer involves performing an analysis of the individual properties associated with semantic and QoS service composition in such a distributed system. This can further be divided and defined into the following categories:

Semantic Level Adaptive Composition: The service provider specifies the possible alternative adaptations of semantics, in the forms of pre- and post-condition tuples, along with the default specification of semantics. The service would then change between these adaptation alternatives based on its environment. When the composing agent is aware that a service involved in the composition process has changed (and becomes invalid in the composition in its current form), it will follow a defined set of algorithms (e.g. domain independent algorithms provided by the general framework, or domain independent algorithms provided by a domain experts that replaces the invalid service with a new service based on the semantics specification.

QoS Level Adaptive Composition: Adaptive QoS Level Composition requires the examination of existing services in order to identify concrete services that could benefit from adaptive composition. Once the concrete services have been identified it is then the job of the Composition Agent to determine how to properly perform service composition and how to best form a composition algorithm to accomplish this task. Included in this decision making process is the use of any and all heuristics that are generated by the specific system that can be utilized in providing optimal QoS conditions. Prior domain knowledge is also extremely helpful in selecting of critical QoS for the application. In addition, we must also evaluate the application domain, with regards to any domain specific QoS requirements or needs, in which the composition will be used. This decision will play a role into the heuristic algorithms used as part of the underlying QoS level composition. Common approaches to service composition can be found in [20], of which we have made use of a generic version of the algorithm proposed in [20]. By using this approach we will be able to dynamically handle the QoS level composition on an end-to-end basis. When examining the adaptive nature of the composer additional work was needed to dynamically evaluate the concrete services, including the ability to gather and accurately assess the current state of a concrete service in terms of its QoS parameters.

4. IMPLEMENTATION

We implemented the proposed architecture (Figure 3) of the composing agent using the Java programming language. We used the Eclipse IDE to develop, integrate, and test our source code. For execution of our test cases, we made use of the JUnit test suite. Adaptive Service Composition, as indicated above, is implemented at two levels – semantic and QoS levels.

4.1. Semantic Level:

The composition framework carries out the composition of services by only considering the semantic specification of services and the semantic requirements of the user. Services that have similar semantic specifications are represented by one specification, with the specification being

referred to asan abstract service specification. Therefore, in the semantic level composition, the composer generates compositions of abstract services. When the semantics of the services are changed (i.e., services change one abstract specification to a different abstract specification), the composing agent updates the composition by changing the service selection and the service composition patterns.

4.1.1. *Semantic Level Composition without Considering Adaptation*

The semantic composition algorithm that the composer follows is summarized in the following pseudo code.

```
ServiceSpecification searchCompositionService (ServiceRequest request)
        outputs  = request.getOutput();
        return searchCompositionServiceForOutput(outputs, request)
end


ServiceSpecification searchCompositionServiceForOutput (Element outputs, ServiceRequest request)
        outputs = ServiceRepository.getServicesWithOutput(outputs);
        if (service != nil) then      # if there exist one service satisfy the requirement
                return service
        parentService = new ComposedService
        for each output in outputs do
                service = ServiceRepository.getServicesWithOutput(output)
                if (service == nil)
                        return nil;
                parentService.addToComposition(service, "parrellelJoin" );
                new_outputs = service.getInputs() - request.inputs();
                Service new_service = searchCompositionServiceForOutput(new_outputs, request)
                service .addToComposition(new_service, "sequence")
        end
        return service
end
```

Figure 4 : Summary of the algorithms for Semantic Service Composition

In this algorithm (Figure 4), when the client requests a particular service (represented as ServiceRequest), the composer queries the services that match the requested output. If there are services that match with the query, they are selected for the composition. However, the selected services may contain inputs that are unknown to the user (inputs that are not available in the Service Request). In such cases, composer has to recursively find the services that output these unknown inputs. We have used several optimization techniques such as indexing and pruning after some level, and caching previously found service composition patterns. These optimizations in generating the composition are intended to improve the performance of the composed service in execution.

Following the algorithm, the composer does the composition by searching exhaustively for the given output until there is a service, which matches with the user input found. The ServiceRepository finds services for the composer using both exact matching (i.e., the output of the service is exactly same as the output of the request) and relaxed matching (i.e., the output of the service is ontologically a sub-class of the output of the request). For relaxed matching, the repository uses the knowledge of domain ontology. Domain ontology contains the relationships among the entities in a particular domain and developed by domain experts. Our framework use Web Ontology Language [19] to represent the ontology.

The following sections describe how the algorithm mentioned in the Figure 4 is applied to a set of services to form an indoor tracking system called eDOTS. The eDOTS is explained in detail in the Section 5 (Experimentation & Results).

The list of services in the eDOTS and their semantic specifications are shown in the Table 1.

TABLE 1
THE LIST OF SERVICES OF THE EDOTS WITH THEIR SEMANTIC SPECIFICATIONS

Service Name	Service Description	Pre-condition	Post-Condition
Camera Service	Provide a video stream	-	[Output = CameraReading]
Marker Repository	Repository of Markers	[Input=Item.Id]	[Output=Item.Marker]
UFilter Service	Filter the video stream based on a marker	[Input=SensorReading]	[Output= FilteredSensorReading]
Kalman Service	Fuse the video streams to	[Input=FilteredSensorReading]	[Output=Item.Position]
	derive the position of a marker		

The domain ontology used in performing relaxed matching is shown in the Figure 5. This ontology is created using tools, such as Protégé OWL editor[11], and Hermit OWL Reasoner[7].

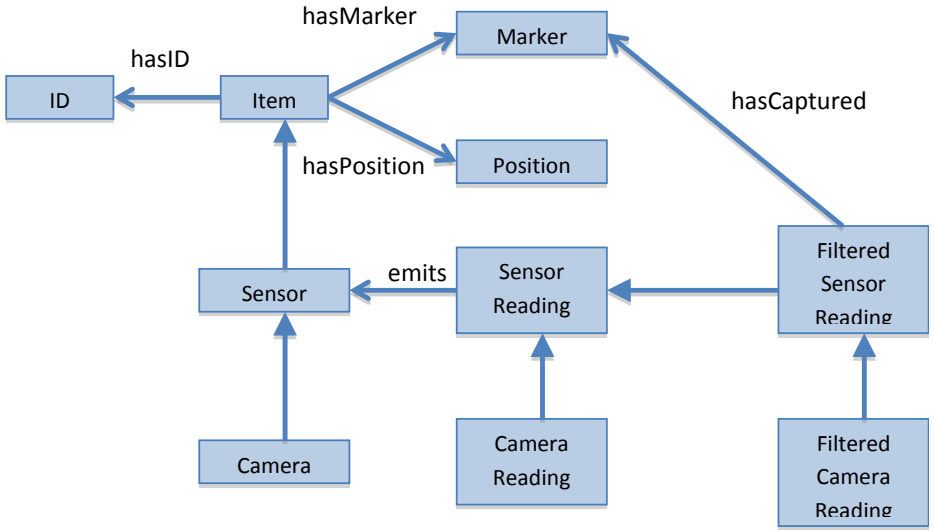

Figure 5 : Domain Ontology associated with the eDOTS system

Applying the algorithm shown in Figure 4 to the services mentioned above will result in the service composition tree shown in Figure 6. In the Figure 6, services in the same level in a branch execute in a parallel or sequential manner (based on the pattern name shown in the parent of the branch) to form the composed service in the parent of the branch.

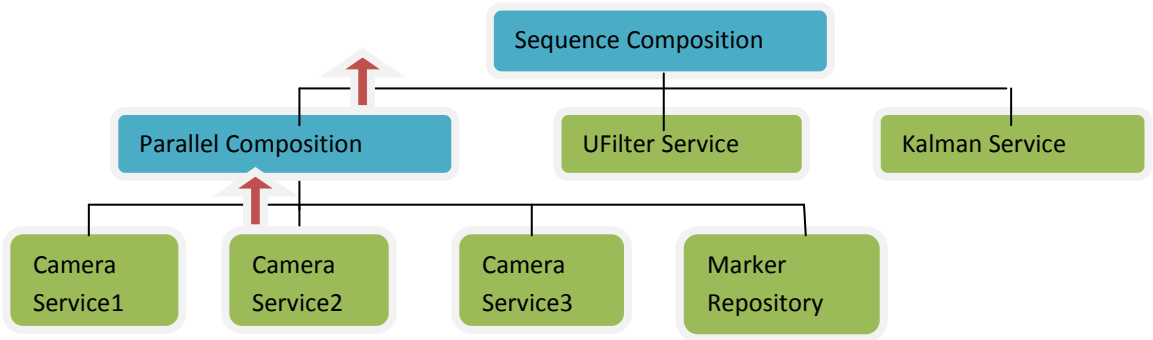

Figure 6 : An example instance of Service Composition

4.1.2. Semantic Level Composition Considering Adaptation

When a service changes, a composition that includes that service also has to be updated or all the compositions that include that service have to be updated to make sure that the composed service still satisfies the client's requirements. For that, reevaluating the whole composition from the start is a big overhead. Therefore, it is necessary to perform heuristics that optimize the adaptation of the composition process.

The heuristics we used in adapting the composition aims at replacing the changed service with another suitable service or an appropriate composition pattern at the same level of the composition tree. If that is not feasible, we traverse one level above in the composition tree (along with the changed service) and try to replace the parent with a suitable service or a pattern. This process continues up the composition tree until we are able to find a suitable replacement or we reach the root of the composition tree. If we reach the root, it indicates that we need re-compose the entire system from scratch, thus, causing a significant overhead. The following pseudo-code (Figure 7) describes the overview of the heuristic algorithm we used in the semantic composition re-evaluation.

Below we describe its usage in the context of the eDOTS, as an example of the execution of this semantic composition adaptation algorithm.

In the eDOTS when the semantic of camera service is changed, the above algorithm performs the following steps. The arrows in the Figure 6 indicate the direction of the update of the composition that is described below.

1. Query the Service repository for a service that matches old semantics of a camera service. If a service (a single or composition) is found, replace the existing camera service by this newly discovered service and complete the adaptation process. Else, continue to the next level.

```
# This function returns true, if a change is required in the specification of a parent service for a given
# service specification
boolean reavaluateComposition (ServiceSpecification composedSpecification)
        if not composedSpecification.isComposition() then          # this is an individual service
                ServiceSpecification individualService = composedSpecification

                if not indivdiualService.changed() then          # first try to replace the individual service
                        ServiceRequest request = individualService.getServiceRequest()
                        ServiceSpecification newService = searchCompositionService(request)
                        if newService = nil then                 # no replacement found
                                return true                      # report to the parent about the failure
                        else                                     # replacement found
                                individualService.replaceTo(newService)
                                return false                     # report to the parent about the success
                        end
                else
                        return false                             # no update necessary
                end
        else                                                     # processing composed services
                for each service in composedService do
                        boolean changedRequired = reavaluateComposition (service)
                        if changedRequired then
                                ServiceRequest request = composedService.getServiceRequest()
                                ServiceSpecification newService =
 searchCompositionService(request)
                                if newService = nil then                 # no replacement found
                                        return true                      # report to the parent
# about failure
                                else                                     # replacement found
                                        composedService.replaceTo(newService)
                                        return false                     # report to the parent
# about success
                                end
                        end
                done
        end
        return false
end
```

Figure 7 : Summary of the algorithm for Adaptive Semantic Service Composition Re-evaluation

2. Query the Service repository for a service that matches semantics of parallel composition service. If a service (a single or composition) is found, use it to replace the existing parallel composition service and complete the adaptation process. Else, continue to the next level.

3. Query the Service repository for a service that matches semantics of the root service. If a service (a single or composition) is found, use it to replace the entire eDOTS and complete the adaptation process. Else, report to the user that an adaptation is not feasible with the changed camera service.

These updated are reflected at runtime in the generated service composition. The number of levels that the search process will travel depends on the height of the composition tree, which in turn will be decided by a specific application. Therefore, the composition framework allows the user to configure the order of the algorithms, and add new adaptation algorithms based on their domain knowledge, such as the knowledge of which services change more frequently and which group of services are better candidates to replace another group of services. Use of such knowledge in custom adaptation algorithm will reduce the overhead of searching exhaustively for the replaceability by the framework.

4.2. Quality of Service (QoS) level

As part of the multi-level specification, there is a need to determine the QoS-related attributes that are appropriate for a given distributed system so that instrumentation can take place for handling changes related to the values of these attributes. Table 2 shows the QoS attributes for the camera service associated with the eDOTS.

For the QoS attributes, we would need to use heuristics to aid in the composition process. Because of this, the heuristic algorithms would need to be dynamic in nature to handle the possibility of change, i.e., the composing agent would need to maintain a history of execution and learn and adjust from it. Therefore, it was necessary to evaluate prevalent algorithms and select a specific one for our prototype. After a careful study of prevalent choices, we selected the Cross Entropy Heuristic algorithm. In this algorithm, if the number of available choices for services is high then the number of solution groups that are considered in iteration should also be high in order to get the optimal result. This algorithm performs in a dynamic manner such that if optimal result requirements are not necessary then the grouping constraint could be relaxed in an effort to provide a higher degree of feasibility rather than optimality. Because of this ability, we selected the use of this algorithm in the QoS-based composition process. In our previous work [5], we have provided a generic modified version of the Cross Entropy Heuristic algorithm – shown in Figure 8 – and used in the adaptive agent.

TABLE 2
SAMPLE CAMERA SERVICE QOS ATTRIBUTES

	Response Time	**Accuracy**	**Clock Drift**	**Resolution (Pixels)**
CameraService	0 – 30 ms.	0 – 500 mm	0 – 15 ms.	320 x 240

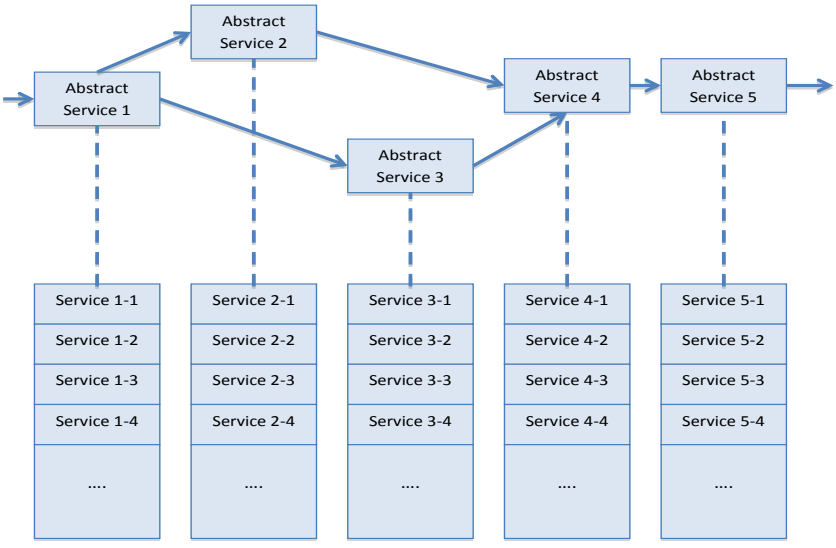

Figure 8 : Generic Cross Entropy Heuristic Algorithm [5]

An example of the adaptive composition between the Camera Services and the Kalman Filter Service is shown in Figure 9. Here the composing agent must make a decision about selecting an appropriate service in the presence of a change based upon the QoS attribute of response time.

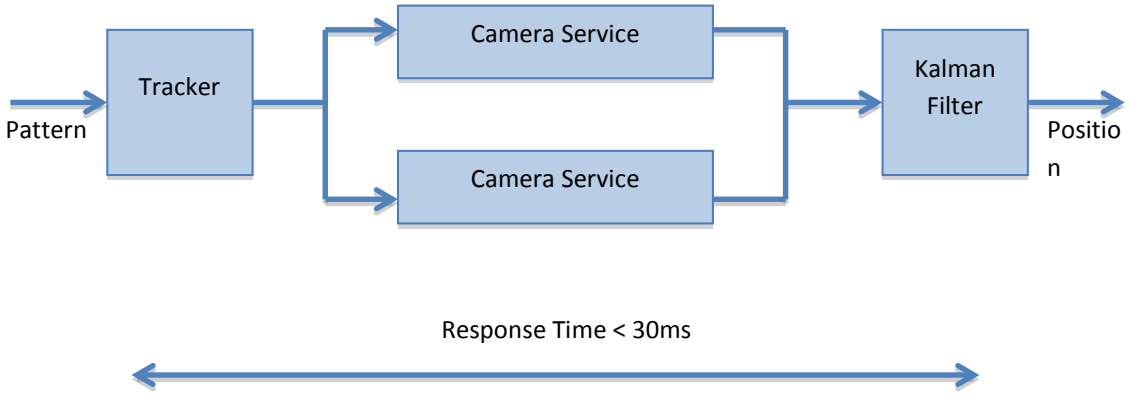

Figure 9 : Adaptive QoS Composition

5. EXPERIMENTATION & RESULTS

We empirically evaluated the composing agent and associated framework using the eDOTS as a case study. As indicated earlier, the eDOTS is used to track the position of a moving object using a set of sensors. The eDOTS is expected to provide QoS requirements, such as the response time and the position accuracy. Therefore, the eDOTS is an ideal candidate to test the applicability of the proposed techniques. The services available for the eDOTS are shown in Table 3.

TABLE 3
eDOTS SERVICES

Service Names				
Camera Service	UFilter Service	Pattern Repository Service	Kalman Filter Service	Fusion Service

Each of these services registers a service level contract upon registration with the JINI Lookup Service. The JINI lookup service is responsible for the task of service discovery. A sample of these service level contracts is shown in Figure 10.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <Camera03>
3    <QoS>
4      <resolution>320x235</resolution>
5      <frameRate>30</frameRate>
6      <clockDrift>12</clockDrift>
7    </QoS>
8  </Camera03>
9
10
```

Figure 10 : Sample Camera QoS Level Contract

In the sample contract shown in Figure 10 we record the name of the specific sensor as well as the specific QoS attributes associated with said sensor. These attributes include Resolution, Frame Rate, and Clock Drift. This contract is written in standard XML formatting. We now will provide our experimental results and our analysis at both the Semantic and QoS levels.

5.1. Semantic Analysis

The adaptive composing activity, as carried out by the composing agent, at the semantic level of the eDOTS uses Camera Services, Marker Repository, UFilter Service, and Kalman Service. The composition of the eDOTS with the default specifications (i.e., without any adaptation) is shown in Figure 11. And this will be the composition configuration when the services are not changing and referred to as main composition configuration. In this composition, the semantics of the services are not expected to change during its execution life-cycle, therefore the semantics of the composition also will not change. Therefore, the static composition is not required to be adaptive.

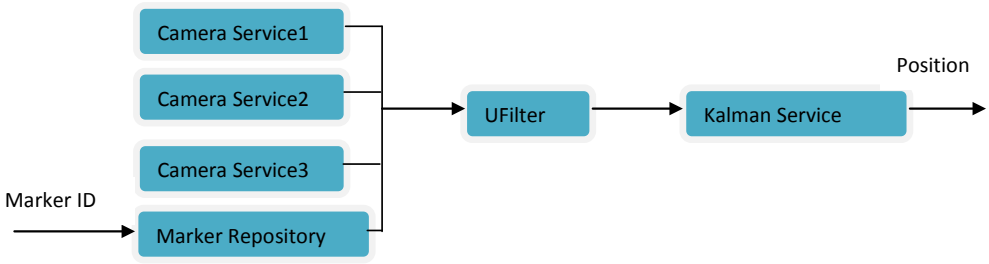

Figure 11 : Static Semantic Composition (Main Composition Configuration)

In this case study, we simulated two adaptations:

1. The Camera Service provides the filtering of the marker themselves as an alternative semantics.
2. The Kalman Service does the filtering before the fusion as an alternative semantics.

TABLE 4
THE SEMANTICS OF THE MAIN AND ALTERNATIVE SPECIFICATIONS IS SHOWN IN THE

Service	**Main Specification**		**Alternative Specification**	
	Pre-Condition	Post-Condition	Pre-Condition	Post-Condition
Camera Service	-	[Output=VideoStream]	[Input=Item.Id]	[Output= FilteredCameraReading]
Kalman Service	[Input= FilteredSensorReading]	[Output=Item.Position]	[Input= {SensorReading, Item.Id}]	[Output= Item.Position]

The changes in the camera services and Kalman Service are simulated using a timer. Because of which both the services are changed to Main and Alternative specifications at random times within a 20 second interval. This results into three composition alternatives as shown in Figures 12, 13, and 14.

5.1.1. Composition Alternative1:

Composition alternative1 occurs, when only the camera services are changed to the alternative specification. The Figure 12 shows the composition diagram, when all the camera services are changed to their alternative specifications.

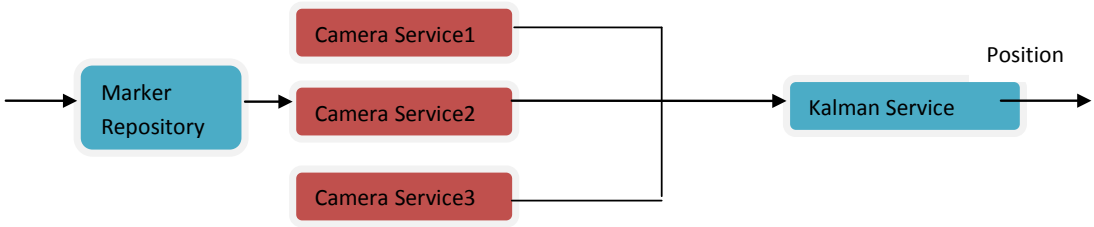

Figure 12 : Adaptive Semantic Composition: Composition Alternative 1

5.1.3. Composition Alternative 3:

Composition Alternative3 occurs, when both the Camera Service and Kalman Service are changed to their alternative specifications as shown Figure 14.

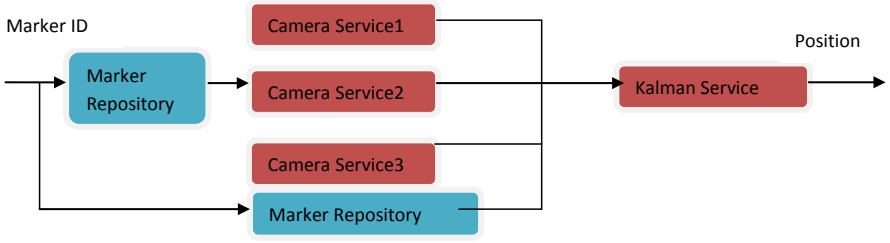

Figure 14 : Adaptive Semantic Composition - Composition Alternative3

We measure the roundtrip time in each of these three composition configurations.

1. Scenario1: Only the camera services alter its service specification periodically within the 20s interval. This will change the composite system between the main composition configuration (Figure 11) and the composition alternative1 (Figure 12).

2. Scenario2: Only the fusion service alter its specification periodically within the 20s interval. This will change the composite system between the main composition configuration (Figure 11) and the composition alternative2 (Figure 13).

3. Scenarion3: Both the camera services and the fusion service alter their service specifications periodically within the 20s interval. This will change the composite system between the main composition configuration (Figure 11), composition alternative1 (Figure 12), and composition alternative2 (Figure 13) and composition alternative3 (Figure 14).

The performance of the composite system generated in these three scenarios is compared to evaluate the adaptations described in the three scenarios above.

The round trip times of the service call in each of the above scenarios are shown in the following graphs (Figure 15, 16, and 17).

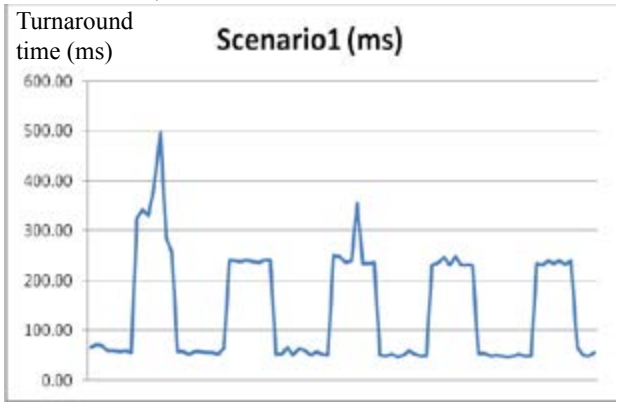

Figure 15: Turnaround time of the tracking system for scenario1 for sequence of requests

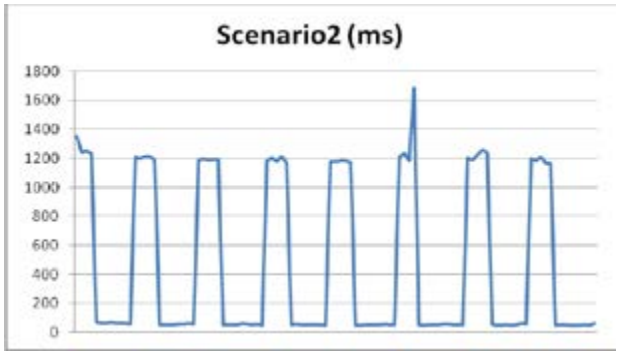

Figure 16: Turnaround time of the tracking system for s cenario2 for sequence of requests

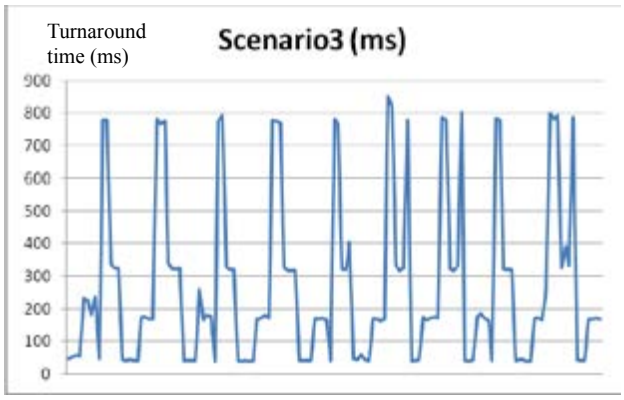

Figure 17: Turnaround time of the tracking system for scenario3 for sequence of requests

The above graphs show the variations of the turnaround times at sequence of requests to the adaptive composed service. These variations are due to the re-evaluation of the composition of abstract services. The semantic composition algorithm takes different time durations for different compositions (Main, alternative1, alternative2).

These graphs show that re-evaluation of composition (caused by changes in some services) take different time based on the location of the adapted service in the composition. For example, changes only in the Camera service (Figure 15) cause less re-evaluation time than the re-evaluation time caused by the changes in the in Fusion service (Figure 16). A possible reason for this behavior can be that as the Fusion service is situated near the outputs of the composed service (the root of composition tree), it causes the re-evaluation of the major part of the composition tree; whereas the Camera service is situated near the inputs of the composition, and therefore, the changes in Camera service has comparatively lesser impact on the composition tree. When both the services change, as the whole composition tree has to change, rather than re-evaluating the partial composition tree, the Composition Agent re-evaluates the complete composition, which causes a lesser peak of turn-around time in Scenario3 than Scenario2.

5.2. QoS analysis

In an effort to validate our results, we needed to examine the cause behind the additional overhead created as a result of our experimental implementation. Therefore, an experiment was designed in an effort to better evaluate and examine the performance of the eDOTS in both the eager and lazy compositions while comparing the overall end-to-end response time (EERT) of the system in the presence of no adaptive service composition.

This experiment was conducted using four Windows XP Pentium 4 machines running the eDOTS. The system times were written to a text file for future analysis. This experiment was conducted over the period of ten minutes of tracking. A single pattern, HiroPatt [9], was used for tracking purposes. This tracking pattern was maintained in the repository. The pattern was placed within the range of the four Cameras in an effort to maximize the tracking process. Table 5 shows the specific QoS attributes of each of the Camera Services involved in the experimentation.

TABLE 5
SENSOR QOS ATTRIBUTES UTILIZED DURING EXPERIMENTATION

Sensor Name	Resolution	Frame Rate	Clock Drift	Rank
Camera03	320 x 240	30	7.6	1
Camera30	640 x 360	15	7.6	2
Camera04	320 x 240	30	7.6	1
Camera40	640 x 360	15	7.6	2

An explanation of the QoS Attributes of each of the Camera Services is as follows:

- Resolution: This attribute is specific to the camera services and is specified by the user. This attribute is limited by the hardware of the specific device, but has the ability to be adjusted within a set range. Therefore, this can be varied throughout the course of a service's execution.
- Frame Rate: This attribute provides the number of video frames per second that the camera itself can provide as part of the video stream. This attribute can be specified by the user but is limited by the hardware performance of the device itself.
- Clock Drift: This attribute provides the expected clock drift that the Camera Service may encounter as part of its execution. This average (7.6) is taken from domain knowledge [3, 4, 5, 6] as the average clock drift found during past experimentation with the eDOTS. This attribute will undergo significant fluctuations during the course of execution and thus, is extremely dynamic.
- Rank: This attribute is initialized to a default value but during the course of the service execution is determined by the Camera Services themselves. This attribute is defined as follows: Rank 1 – this camera provides the highest QoS available within the system, Rank 2 – this camera provides adequate QoS in its ability to track an object. The first two attributes (Resolution, Frame Rate) remain constant during the course of object tracking while the latter two (Clock Drift and Rank) will change over time and be subject to system performance.

Table 6 shows the effects of both eager and lazy adaptive QoS composition within the eDOTS when compared to a situation without the adaptive composition.

TABLE 6
ADDITIONAL OVERHEAD (MILLISECONDS)

QoS Composition Type	Average Time	Maximum Time
Lazy	45.8	141.0
Eager	89.3	266.0
Without Adaptive Composition	39.1	128.0

As shown in Table 6, the overhead introduced in the lazy evaluation is minimal but is substantial in the eager evaluation process. We believe this is the case because the lazy evaluation is passive and does not introduce any additional communication, which is extremely costly in the composed system. In addition, the time in the lazy composition process is similar to that which was found in the existing eDOTS without any adaptation. The minor difference between these two situations is due to the attribute matching (Resolution, Frame Rate, Clock Drift, and Rank) as well as the time to create the composed version of the eDOTS. In the eager evaluation process, we noticed that the overhead is much higher due to the additional communication required to check the current status of the services. This overhead may be too much for certain application domains to withstand and a possible future extension would be to design a hybrid model for the dynamic adaptation, which can switch between the lazy and eager alternatives as needed.

A second set of experiments was conducted in order to evaluate the effectiveness of the service composition process. In this set of experiments, we chose to vary a subset of the QoS parameters of various services in a random manner. We then examined the overall effectiveness of the composing agent with respect to the actual modifications of the concrete services. Effectiveness was gauged in two parts: first with the service request being met and providing a high degree of service accuracy, and second with respect to the real-time timing constraints of the eDOTS. In this experiment, we used the Kalman Filter service as the means for providing data fusion and calculated the estimated error based upon the physical measurements of the tracking marker. This experiment was conducted on two different machines, each with two cameras connected to it. Each of these cameras had the same QoS attributes as

TABLE 7
SYSTEM ACCURACY WHEN USING ADAPTIVE COMPOSITION

QoS Composition Type	**Lazy**	**Eager**	**None**
Accuracy (mm)	101.2	81.3	88.8
Number of Services	4	2	4

As show in Table 7, the overall accuracy of the composed system is maintained or improved while using the adaptive composition. In both the lazy and non-adaptive schemes, we see a similar outcome in terms of both accuracy and the number of services utilized in the tracking process. For the eager evaluation process, because of the adaptive composition, we see that only two services have been selected as part of the data fusion process. These two services have been selected and composed based upon their dynamic attributes, thus ensuring the QoS as desired by the client. Through this analysis, while the eager evaluation provided an improved accuracy, due to the dynamic composition, it also caused additional overhead, which may or may not be acceptable. This is a tradeoff that would need to be weighed by the application domain expert when using the eager scheme in the composition process. In either case, our results demonstrate the feasibility of adaptive composition within a realistic distributed system application. As a result, we have shown that through the use of Adaptive QoS Composition we can provide a high level of accuracy while maintaining the rigorous constraints of a real-time tracking application.

6. CONCLUSION & FUTURE WORK

Based on the proposed composition schemes and associated empirical validation, we can draw the following inferences:

- Adaptive Composition is possible within a practical distributed system such as the eDOTS using the framework described in this paper.
- Additional overhead is introduced as part of the composition process, this is due to the extra processing of the composing agent to re-evaluate the composition and the extra communication between composing agent, repository, and services in re-locating and using services from the available services.
- It is comparatively difficult to predict the turnaround time of the system such as eDOTS when services change semantically as it depends upon the frequencies of changes in different services.

In our experiment, we simulated the adaptive behavior of services in the eDOTS application. Practically, when eDOTS is integrated with sensors of a mobile phone such as camera, GPS, and WIFI tracker, as these services adaptively change their behavior/specifications based on the environment such as battery level, internet connection strength, the application that has composed of those services have to be changed accordingly. Our adaptive composition framework attempts to hide the complexity of such changes in the compositions by providing necessary abstractions. We believe that through our analysis that Adaptive Composition is a practical solution for dynamic multi-level distributed systems.

- The future work related to the research on adaptive composition includes,
- Studying and implementing adaptive composition in syntax and synchronization contract levels.
- Experimenting with different heuristic algorithms to improve the performance of the adaptive composition.
- Studying of tight integration of domain knowledge to derive alternative specifications of services, alternative requirements of clients and alternative composition of systems in adaptive environments.

REFERENCES

[1] A. Beugnard, J.M. Jézéquel, N. Plouzeau, D. Watkins, "Making Components Contract Aware," *IEEE Computer,* Vol. 13, no. 7, 1999.

[2] A. Bracciali, A. Brogi, C. Canal, "A formal approach to component adaptation," *J. System Software,* Vol. 74, pp. 45-54, 2005, DOI=10.1016/j.jss.2003.05.007 http://dx.doi.org/10.1016/j.jss.2003.05.007.

[3] B. Zhang, Y. Shi, X. Xiao, "A Policy-Driven Service Composition Method for Adaptation in Pervasive Computing Environment," *Comput. J.*, Vol. 53, no. 2, pp. 152-165, 2010. DOI=10.1093/comjnl/bxm103 http://dx.doi.org/10.1093/comjnl/bxm103.

[4] C. Dabrowski, K. L. Mills, S. Quirolgico. "A Model-based Analysis of First Generation Service Discovery Systems," *NIST Special Publication*, pp. 500-260, 2005.

[5] D. Gamage, R. Gamage. "QoS Aware Service Composition Framework," *Indiana University Purdue University Indianapolis – Computer Science Department*, 2011.

[6] D. Hemer, "A formal approach to component adaptation and composition," in *Proceedings of the Twenty-eighth Australasian conference on Computer Science ,* Australia*,* Vol. 38, pp. 259-266, 2005.

[7] Hermit OWL Reasoner, http://hermit-reasoner.com/, 2011.

[8] J. Camara, G. Salaun, and C. Canal, "Run-time Composition and Adaptation of Mismatching Behavioural Transactions," in *Proceedings of the Fifth IEEE International Conference on Software Engineering and Formal Methods (SEFM '07)*, IEEE Computer Society, Washington, DC, USA, pp. 381-390, 2007. DOI=10.1109/SEFM.2007.35, http://dx.doi.org/10.1109/SEFM.2007.35, 2007.

[9] Kato H., Billinghurst M., "Marker Tracking and HMD Calibration for a video-based Augmented Reality Conferencing System," pp. 85-94, San Francisco, 1999.

[10] P. K. McKinley, S. M. Sadjadi, E. P. Kasten, and B. H. C. Cheng, "Composing Adaptive Software," *Computer*, Vol. 37, no. 7, pp. 56-64, 2004. DOI=10.1109/MC.2004.48, http://dx.doi.org/10.1109/MC.2004.48, 2004.

[11] Portege team, "The Portege Ontology Editor and Knowledge Acquisition System," http://protege.stanford.edu/. 2011.

[12] R. Rybarczyk, R. Raje, M.Tuceryan. "A Heterogeneous Indoor Tracking System," *Indiana University Purdue University Indianapolis – Computer Science Department*, 2012. TR-CIS-0125-12, 2012.

[13] R. Rybarczyk, R. Raje, M. Tuceryan, "Enhancing a Distributed Tracking System," in *Proceedings of 3rd International Joint Conference on Information and Communication Technology (IJcICT),* Mumbai, India, pp. 7, 2011.

[14] R. Rybarczyk, "e-DTS 2.0 – A Next-Generation of a Distributed Tracking System," *Master's thesis, Purdue University*, 2010.

[15] R. Rybarczyk, R. Raje, and M. Tuceryan, "e-DTS 2.0 - A next-generation of a Distributed Tracking System," in *Proceedings of the International Conference on "On Demand Computing" (ICODC-2010)*, Bangalore, India, 2010.

[16] R. Aschoff, A. Zisman, "QoS-Driven proactive adaptation of service composition," in *Proceedings of the 9th international conference on Service-Oriented Computing (ICSOC'11),* Gerti Kappel, Zakaria Maamar, and Hamid R. Motahari-Nezhad (Eds.). Springer-Verlag, Berlin, Heidelberg, pp. 421-435. DOI=10.1007/978-3-642-25535-9_28 http://dx.doi.org/10.1007/978-3-642-25535-9_28, 2011.

[17] S. Phatak, "Multilevel Specification for Adaptive Services," *Master's thesis, Purdue University*, 2009.

[18] The OWL API, http://owlapi.sourceforge.net/, 2011.

[19] W3C OWL Working Group, "Web Ontology Language (OWL)," http://www.w3.org/2004/OWL/. 2007.

[20] Y-S. Luo, Y. Qi, D. Hou, L-F. Shen, Y. Chen, X. Zhong, "A novel heuristic algorithm for QoS-aware end-to-end service composition," *Computer Communication*, 2010. doi:10.1016/j.comcom.2010.02.028.

ABOUT THE AUTHORS

Dimuthu U. Gamage: Dimuthu is a graduate student and currently reading for his PhD in computer science at the Department of Computer and Information Science, at IUPUI. Dimuthu's current research focuses on quality aware and trust-based service composition and generative programming in distributed software systems. Dimuthu is a member of the software engineering and distributed systems (seds) group at IUPUI.

Ryan Rybarczyk: Ryan is a graduate student and currently reading for his PhD in computer science at the Department of Computer and Information Science, at IUPUI. Ryan's research interest includes Distributed Systems and Software Engineering with an emphasis on Pervasive Computing and its applications with respect to indoor tracking systems. Ryan is a member of IEEE, ACM and the software engineering and distributed systems (seds) group at IUPUI.

Dr. Rajeev R. Raje : Dr. Raje is a Full Professor and Associate Chair in the Department of Computer and Information Science at Indiana University-Purdue University Indianapolis (IUPUI). Dr. Raje's research interests are in the fields of distributed computing, programming, object/ component-based software systems, and software engineering. Prof. Raje is a member of the software engineering and distributed systems (seds) group at CIS, IUPUI. Prof. Raje is a member of the ACM and IEEE.