QoS Centric Functional Adaptation for composite Services in SOA

P. Gangadhara
M.Tech(Software Engineering), Computer Science Department, JNTUCEA, India.
gangadhara115208@gmail.com

Abstract

Present Service oriented architecture (SOA) used in many globally distributed huge software systems. In such systems networks have collaborative environments, and they are in constant fluctuations, concepts like interactions and task delegation executions need continuous readjustments, requiring flexible and context based interaction models. Software engineering has to be face that challenge that how to control such systems powerfully and successfully. Sophisticated adaption techniques required for improving the collaborations in those systems. To overcome the problems that are not solved in self-adaptive managed SOA based systems, dynamic adaption methods required to deal with composite services along with peer-to-peer, distributed architectures. An approach suggested for the systems with dynamic adaptive composite services based on widespread protocol, and is based on three layer reference model for such systems uses a dialogue protocol for information distribution and decision making. Proposed system objective is to offer dynamic adaption in composite web services with asynchronous execution to achieve structural and global QOS requirements.

1. Introduction

Software engineering has a major challenge that how to manage, large and complex computing systems with high amount of physical distribution. Some application domains are examples for such systems described in [4]. The use of autonomic features suggested as a possible action for overcome these problems shown in [12], [8]. The autonomic research model allows the system to quickly self configurable in response to the variations of operating conditions. Scalability and robustness criterion impose absolutely distributed solutions to achieve these autonomic capabilities shown in [6]. The dynamic nature and having more number of elements of these systems solve scalability and complexity issues well beyond conventional cases.

Objective of this paper is to offer distributed solutions to manage large and highly dynamic distributed systems. So the focus is on the issue of developing a dynamic adaptive method amongst composite services offered by different peers, whose goal is to fulfill required and offered services, in addition to this the system operates under different requirements concerning quality of services (QOS) (e.g., dependency, performance, cost). Therefore the multi-agent system is suitable to deal with different functionally feasible composite services to meet structural and global QOS requirements.

According to the axioms in the distributed systems [2], the solution to be proposed in order to reach QOS goals the proposed solution describes some properties which are mentioned below.

- If the external control absents the self-configuration procedure starts internally.
- In runtime the system able to adapt itself to changes continuously.
- In the absence of central control the composition among peers maintained through local decision making with information distribution to improve decision making process.

The system that design to adopt the architectural procedures depend upon the construction consists of three layers for self-adaptation of services proposed and applied to distributed systems in [10] and [5]. To support distributed decision making, information distribution the intermediate layer architect according a gossip protocol. The solution is able to resolve composite service grouping and whole system restructure itself to deal with failures of nodes.

2. System Model

In this section the system’s model and terminology has defined to use in this paper. To achieve simplicity and readability, it is assume that every peer hosts as a single service. So that it will use same name for both a service and the peer with a minor misuse of notation. The system that regard contains variety of
services(S) of distributed composite Services (N) where S = \{S₁, ..., Sₙ\}, the services can be located anywhere, by using network the interactions can take place. Every service provides an interaction to serve well to the clients requests, these interactions spread the functionalities. Basically, a service S can be represented by using a tuple S = (t, δ, u, Pred, Succ).

- The interface types provide is t \in T, and the assumption is T ≥ 1 different services, T = \{1, ..., T\}. Set of input/output types basically defined by each service type of S, provides the different services.
- Dependencies of S area set represent by δ \subseteq T, the set δ is required dependency types in S.
- Service S utility is considered as a measure of QOS (e.g., execution time, reliability, and cost).
- Set of services bounded to S is Pred \subseteq S to resolve some of its dependencies.
- Set of other services of S is currently bound to resolve some of their dependencies are Succ \subseteq S.

A service \( S \in S \) denotes \( t [S], δ[S], u[S], Pred[S], Succ[S] \), the service type, set of dependencies, local utility and the sets Pred, Succ of service S. A composite service group G is an acyclic graph \( G = (S, E) \) Where \( E \subseteq S \times S \) is the set of dependencies. A directed edge \((Sᵢ, Sⱼ) \in E\) denotes that \( Sᵢ \) uses \( Sⱼ \) to resolve one of its dependencies. A service S is fully resolved in given composite services group G either

- If S has no dependencies (\( δ[S] = \emptyset \)); or
- For all \( d \in δ[S] \), there exists a service in S, that \( S' \in Pred[S] \) of type \( d \), i.e., fully resolved itself.

In composite services group G, S is not fully resolved, but partially resolved.

3. Evaluation of Complex Utility

Complex utility function \( U(S) \) is important part in this model, building and maintain fully resolved services group to maximize the complex utility of each composite services is the proposed system goal. \( U(S) \) is a function recursively calculates the complex utility of services in S defined in terms of \( u(S) \). If S is not fully resolved it should be \( U(S) = -\infty \)

**Complex utility based on reliability:** For the services S its reliability is the probability that for a certain service request S completes correctly its tasks. The \( u[S] \) is the internal reliability of \( S \). It is the probability when S executed, there is no internal failure. When S executed during execution \( n(S') \) is concerned, \( n(S') \) depend on \( t[S] \) type.

\[
U_r(S) = \left\{ \begin{array}{ll}
u(S) \times \prod_{S' \in Pred[S]} u(S') & \text{if } S' \in Pred[S] \\
\infty & \text{otherwise}
\end{array} \right.
\]

Overall reliability of groups of composite services S is \( U_r(S), (Pred[S] = \emptyset, the product is set to 1, therefore } U_r(S) = u[S] \).

**Complex utility based on cost:** The average cost of a service in S is the average cost of single execution of S. There are two cases distinguished. In the first case if S is a service that represents \( u[S] \) that is negative, if utility is high it is better metric. The cost based complex utility of S is \( U_c(S) \) evaluated as below.

\[
U_c(S) = \left\{ \begin{array}{ll}
u(S) + \sum_{S' \in Pred[S]} U_c(S') & \text{if } S' \in Pred[S] \\
\infty & \text{otherwise}
\end{array} \right.
\]

The Sum is set to zero and \( u[S] = U_c(S) \) if \( Pred[S] = \emptyset \). In the second case for the use of a service a fixed cost paid to adapt a model of the flat cost, the flat cost is \( U_f(S) \) evaluated as follows.

\[
U_f(S) = \left\{ \begin{array}{ll}
u(S) + \sum_{S' \in Pred[S]} U_f(S') & \text{if } S' \in Pred[S] \\
\infty & \text{otherwise}
\end{array} \right.
\]

**Complex utility based on response time:** The overall average time taken to fulfill service request in S, the service S average time is \( -u[S] \), the \( u[S] \) is negative, so the complex utility \( u[S] \) is better metric, the average response time of a service in S, and it can be computed as \( U_t(S) \).

\[
U_t(S) = \left\{ \begin{array}{ll}
u(S) + \sum_{S' \in Pred[S]} U_t(S') & \text{if } S' \in Pred[S] \\
\infty & \text{otherwise}
\end{array} \right.
\]

**Structural requirements:** The structural requirements are also important to the composite services groups, as well as the QOS requirements. These requirements concern global properties local properties. In global properties to enforce the global constraints in a system is difficult. Local constraints considered that S can resolves maximum number of dependencies means that most \( D_{max} \) uses the S, other services resolve their dependencies. Local utility for the first \( D_{max} \) services is \( u[S] = 1 \); and
local utility of a service in S will get \( u[S] = 0 \); if any service enquiring the local utility.

The complex utility of a service in S is \( U_m(S) \) defined as

\[
U_m(S) = \left\{ \begin{array}{ll}
\sum_{S' \in Pred[S]} u_m(S') & \text{for } S \text{ has dependencies}
\\
\sum_{S' \in Succ[S]} u_m(S') & \text{for } S \text{ is single peer}
\end{array} \right.
\]

Then the system driven towards a grouping of services in which no more than \( D_{max} \) used by every service. If S has at most one dependency \( u[S] \) is set to 1, otherwise 0. The complex utility \( U_p(S) \) evaluated as below.

\[
U_p(S) = \left\{ \begin{array}{ll}
\sum_{S' \in Pred[S]} u_p(S') & \text{for } S \text{ has dependencies}
\\
\sum_{S' \in Succ[S]} u_p(S') & \text{for } S \text{ is single peer}
\end{array} \right.
\]

If S has no dependencies \( U_p(S) = 1 \).

**Other metrics**: Besides structural measures or utility based QOS other metrics also defined. The metrics that are to be introduced based on implicit assumption, for a service S is the utility \( u[S] \) depend on number of services that S is bound, \( u[S] \) does not depend on content or size of \( Succ[S] \), for some metric it may appropriate (e.g., reliability, cost) for other works it may be restricted. In terms of a single QOS attribute, above definitions allow the evaluation of complex utility for a single service. The complex utility of multi agent attribute of S for single attributes utility values weighted sum. For a single service S the QOS attributes of M are \( U_i(S) \)......\( U_m(S) \) is computed.

\[
U(S) = \sum_{i=1}^{m} w_i U_i(S).
\]

**4. System Architecture**

The reference model for dynamic adaptive software systems along with three interactive layers architected in [10]. The bottom concerned the adaptation at the level of single component. The middle layer concerns with managing system globally consisting at the lower layer. The upper concerns the definition of new goals and adaptation plans.

**Bottom layer**: At the component control layer each single peer of S responsible for suitable adaptation actions internally. Variations of the value of \( U(S) \) flows towards the middle layer.

**Middle layer**: Building and maintaining the architecture at each peer of \( SPred[S], Succ[S] \) state components are used in change management layer. This layer checks the variations of ability of fulfillment of requirements and also checks whether already determined service group fulfill the notified goal of upper layer.

**Upper layer**: The functional and nonfunctional goals fulfill by this layer and inform to the middle layer about them, there are 2 kinds of non functional goals can be communicate.

- More than goals: A service group is to be determined and its complex utility greater than a specific value.
- Max goals: A group that is to be determined that maximizes complex utility of the certain services.

The upper layer is responsible for implementing a suitable response if any failed notification received from middle layer. A distributed approach is basis for the design of the system that makes the system scalable and robustness. A three layer reference architecture for dynamic adaptive management system shown in below figure.

![Fig. 1: 3 Layer Reference Architecture for Self-Adaptive Management System](image)

**5. System Operations**

This section illustrates system operations how these layers (bottom & middle layer) are construct, maintain and dynamically operate in a completely distributed manner with service groups in present architecture, and these service groups able to satisfy QOS goals. There are two sections; **first section** discus about implementation of gossip based algorithm at the middle layer. **Second section** presents how the operations are implemented at middle and bottom layers.

1. **Gossip based algorithm at middle layer**: From the set of services (S) and the utility function \( U \) the distributed algorithm constructs a service group where each service belongs to the set of services (S) is clearly identified and the value of utility function is gradually increased until it reaches to its maximum value or it reaches to the given threshold value. The gossip based algorithm iteratively resolves the dependencies of services described in [11], [7]. Each peer consists of two protocols in P2P network, and actual service which is placed on top of the peer. To
maintain distributed network over the set of peers, it uses NEWS CAST protocol described in [9]. Each peer uses “a local view “ of the system, which replaces messages among set of peers.

2. Algorithm for wrapping the services run by each peer. The goal of middle layer core algorithm is to discover wrapping of services to its needed dependencies such that the maximization of grouping services can be achieved. This procedure will be continued until it reaches the given threshold value. This section presents the implementation of operations at the bottom layer of architecture at each peer and its execution related to gossip algorithm. The traffic in the message replace can be decreased by sending every service to cache with complex utility and identified and record changes in every service which are currently restrict it. Also the system allows multiple users to request multiple operations which satisfy completely in order to achieve this it uses a flooding algorithm and activates P2P network on basis of requested features.

In some cases the system faces difficulty to find maximum complex utility; because it relies on the root services. So a suitable stopping criteria is needed which basically interrupt the active thread. It uses timestamp mechanism and monitoring the wrapping of group of services. When the things are becomes more complex service grouping does not exist. In that case it use stopping criteria which states the failure rates to maximize QOS goals, it records and identifies each failure rate with the help of stopping criteria to manage goal of the present architecture.

6. Results

Two types of simulation engines are offered by the algorithm discrete-event and cycle-based. To assess Peer-to-Peer protocols, the cycle-based engine is used, in which all interactions happen at different time steps, convergence speed measure is the important metric, and this performance metric is independent of the software, hardware and network infrastructures.

To maximize the complexity utility of each peer consider a system consists of N services with different interfaces T where \( T = \{ 1, \ldots, T \} \). \( N/T \) services types. For each service \( D \) random dependencies are defined. A utility factor \( u[S] \) for each service of \( S \) is assigned to \( (0,1) \) that uniformly distributed. Randomly it choose for every type \( t \in T \). \( N_{opt} \geq 1 \) is set to 1, always optimal service utility of group is 1.

There are two metrics used for performance measure fraction \( R \) and average utility \( U \), the optimal value of \( R \) is 1, by computing fraction of fully resolved groups, when each simulation step is completed. Fully resolved services average utility at step \( t \) is the \( U_t \), its optimal value is 1.

The simulation at different scenarios:

a) Number of neighbors \( K \): The service group average utility is grows slowly, and it becomes stable about a value below the maximum value 1. When optimal services are multiple or number of neighbors increased simulation is improved.

b) Number of Dependencies: when the number of dependencies increases the convergence rate to the optimal utility of the services groups also increases. Maximization of complex utility is the goal of each peer; hence it binds to the services of maximum utility. The gossip protocol has a chance to place dependencies of maximum utility quickly, if peer has more lists of dependencies to replace.

c) Handling failures: The failures occurs for every large set of distributed components present, new peer may connect and some time present peer may crash, to uphold enormous failures chartitably, the gossip-based algorithms has capability.

Figure 2 illustrates the fraction and average utility of fully. A resolved service of each peer consists of number of neighbors for different values of \( K \). For every failure it clearly seen that a spiky fall of the fraction and average utility of resolved components. Within a small number of steps, the algorithm upholds failure nodes and fixed itself near a new and best within small number of steps. The system slows down and become stable about a suboptimal when every node has a limited number of neighbors all most all services fully resolved quickly as shown in the below figure.
7. Related Work

In [5] the strategies that are suggesting based on dynamic service composition, the issue of managing dynamic service composition deals with in the literature.

Decentralized and centralized self-adaptive software systems differences and illustration of decentralized systems importance, to achieve quality requirements in large distributed software systems shown in [6].

A three layer architecture for self-adaptive systems described in [10]. Decentralized systems designs are deeply analyzed in [3]. Some approaches overviews present in [3] and [1].

In [2] a decentralized approach present in which each agent continuously contacts a subset of peers rather whole set to find the organizational composition that completes specific task.

The proposed system goal is to offer dynamic adaption for composite services with asynchronous execution, by which to achieve global QOS in distributed systems. So the focus is on the development of decentralized solution for the distributed systems.

8. Conclusions

A distributed method proposed for the distributed software systems with dynamic adaptation of composite services in this system operations need lot of information at each peer to replace and maintain total number of peers in the system that guarantees the proposed system structural and QOS requirements.

A model of software middleware is to be implemented to understand proposed system algorithm for composite services in different application scenarios and its efficiency should be estimated.

References


