SDP-Implementation in Opnet® v.2

SDP-Based Composite Service Execution in MANETs

- Release Notes -User Manual –Programmer Guide

FRIEDRICH-SCHILLER-UNIVERSITY OF JENA

2010

Preface

Please refer to [1] as a base for this documentation. Based on the SDP implementation in this previous work, we complement here an SDP-based composite service execution layer for MANETs.

Having an idea about the work described in [1] is highly recommended for understanding this documentation.

Table of Contents

INTRODUCTION 3

CONTENTS: 3

WHO IS SUPPOSED TO READ THIS DOCUMENTATION? WHICH PART? 3

WHAT TO DOWNLOAD? HOW TO INSTALL? 3

PART I: RELEASE NOTES: 5

IT REQUIRES: 5

ABILITIES: 5

PART II: USER MANUAL: 6

BASIC CONCEPTS 6

MINIMUM COMPONENTS TO RUN A SDP SIMULATION 6

SELECTING THE REQUIRED STATISTICS 8

NOTES ON THE RESULTS 9

DO NOT FORGET 9

PART III: PROGRAMMER GUIDE 10

SDP PACKETS' STANDARD 10

SDP MOBILE NODE 12

INTERACTIONS AMONG THE MOBILE NODES AND THE REPOSITORY 14

COMPOSITE SERVICE EXECUTION STYLES 14

SEQUENTIAL EXECUTION STYLE 14

PARALLEL EXECUTION STYLE 15

SOURCE CODE FUNCTION PROTOTYPE EXPLANATION AND LISTS-STRUCTURES PROTOTYPES 16

REFERENCES 19

SDP-Implementation in Opnet® v.2

- Release Notes -User Manual –Programmer Guide

Introduction

SDP (the **S**ervice **D**istribution **P**rotocol for MANETS) [1, 2] has been introduced as an approach with many innovative mechanisms to increase the mobile service availability. It is mainly based on a new technique in service replication called "replication-hibernation-restoration". In a previous implementation [1], SDP has been evaluated as a service replication approach for mobile networks. In this version, the previous work has been extended to include consideration for the evaluation of composite service execution processes [3].

Contents:

This documentation consists of 3 parts: I. Release Notes, II. User Manual and III. Programmer guide. In the first part, the composite service execution of composite service requests in MANETs is presented. Part II shows an easy user manual which introduces how to use the delivered software under the case tool of Opnet® Modeler® Wireless. The last part shows an extension for the programmer guide in [1] to make it easy to implement your special extensions.

Who is supposed to read this documentation? which part?

Please refer to the release notes of [1].

What to download? How to Install?

What to download?

Please find and download the SDP-Composite-Execution compressed model file in the Opnet contributed models at "https://enterprise1.opnet.com/tsts/4dcgi/models_searchdialog".

The model compressed file contains the following components:

- A channel matching pipeline:
  - pipeline stage: "dlvr chanmatch75m.ps.c"
- The mobile node:
  - Process model: "CompSerExe SDP Application.pr.m"
  - Node architecture: "CompServExe SDP node arch.nd.m"
- The repository node
  - Process model: "SerEXE CompositeService global_db process.pr.m"
  - Node: "SerEXE CompositeServices global_db.nd.m"

- A sample project:
  - Project1: “SerEXE SDP compositeServices 50.project”

How to install?

Please refer to the same section in the release note of version1 at [1].

Part I: Release Notes:

Please have a look in the previous version of the release notes. This version is the second release version.

It requires:

- The same as in [1].

Abilities:

The new abilities in this release are oriented to investigate the performance attributes of the composite service execution based on SDP in MANETs.

Plus the abilities of the previous release notes[1], it can:

- ❖ Simulate and investigate the SDP performance of the resource competing services.
- ❖ Investigate the composite service sequential and parallel execution plans with expressive attributing measurements of the average response time and the composite success ratio.

Limitations:

- ❖ Considering the previously mentioned limitations in [1].
- ❖ Simulating the loops in the composite service execution plans is not considered in this version and assumed to appear in next versions.

Part II: User Manual:

Basic Concepts

Please have an idea about the basic concepts from the "basic Concepts" section in PartII of the user manual in [1].

Composite Service Execution and Managing the Execution Plans in MANETs

Some required functionalities at the mobile clients cannot be delivered by only one service. These complex client requests need to be posed to many services, each of this services delivers a different functionality. In order to get such a complex functionalities, the mobile clients need to get some knowledge about the available services in their local network partitions from the service discovery components. This information are supposed to be used by the service composition components at these clients to produce the composite service requests. In this implementation, an extended execution layer is added for SDP to enable the mobile clients to manage the generating and controlling the execution plans to meet the posed composite requests. In this report, the services are categorized into atomic and composed services. The atomic services are those which can be hosted individually by one provider. The composite service requests are composed of minor (atomic) requests for the atomic services. All the involved atomic services in our implementations are replicable [2].

Minimum Components to run a SDP simulation

After a successful setup of an Opnet® project with the MANET library, you need to include the following components in the project to be able to run SDP simulations safely and correctly. Please refer to [1].

- Radio Range - Pipeline Stage
- Wireless Domain
- Mobility Config Module
- Repository Node

Wireless Mobile Node

This node is built on the basis of the wireless mobile node architecture of the user manual of [1]. Add as many instances as you want inside your wireless domain. Associate the mobility to the deployed mobile nodes.

How to specify?

Besides the configuration in [1] for the wireless mobile node in the user manual part, we extend the specifications to include the new features of composite service execution processes

As in Figure 1, the new added tab of “application.Composite Services” shapes the needs of the current mobile node of the composite services. Based on the defined service definitions in the global repository node, the current mobile node generates a set of composite service requests. The number of the atomic service requests in the composite service (composite service length) is varying through the network operation time. Each of the mobile nodes starts composing composite requests with an initial request length (using the value of the “Initial Length (Composite Service)” field) then, it generates another longer (the length is incremented using the value of the “Step (next length)(services)” field) request and so on until all of the defined services are included in the longest possible composite service request. After a periodic time interval (defined as a distribution in the “Step (next time)(seconds)” filed in seconds), the current mobile node repeats the previously mentioned composite service requests’ generation cycle. The composite service execution has two extremes of execution styles as mentioned in [3]. These two styles are the sequential and parallel execution styles. In the sequential execution style, all the atomic service requests are performed sequentially. The current mobile node in the sequential style of execution is supposed to pose the service requests for the atomic services one by one and considers the composite request failed if any step of the execution fails. On the other hand, in the parallel execution all the dissolved requests of any composite request are parallel posed to all of the atomic services at the same time. The style of execution is separately supplied for each mobile node through the field of “Style of execution”. The field of “Composite service execution?” simply bypasses the generation of any composite services if it is set to “False”. The field of “Max Execution Time” sets a time constraint (the “-1” value indicates that no constraint is assigned) for any of the generated composite service requests to be executed. If the response time of any of the generated composite requests is greater than this time (in milliseconds), then this request is considered as a failed composite request.

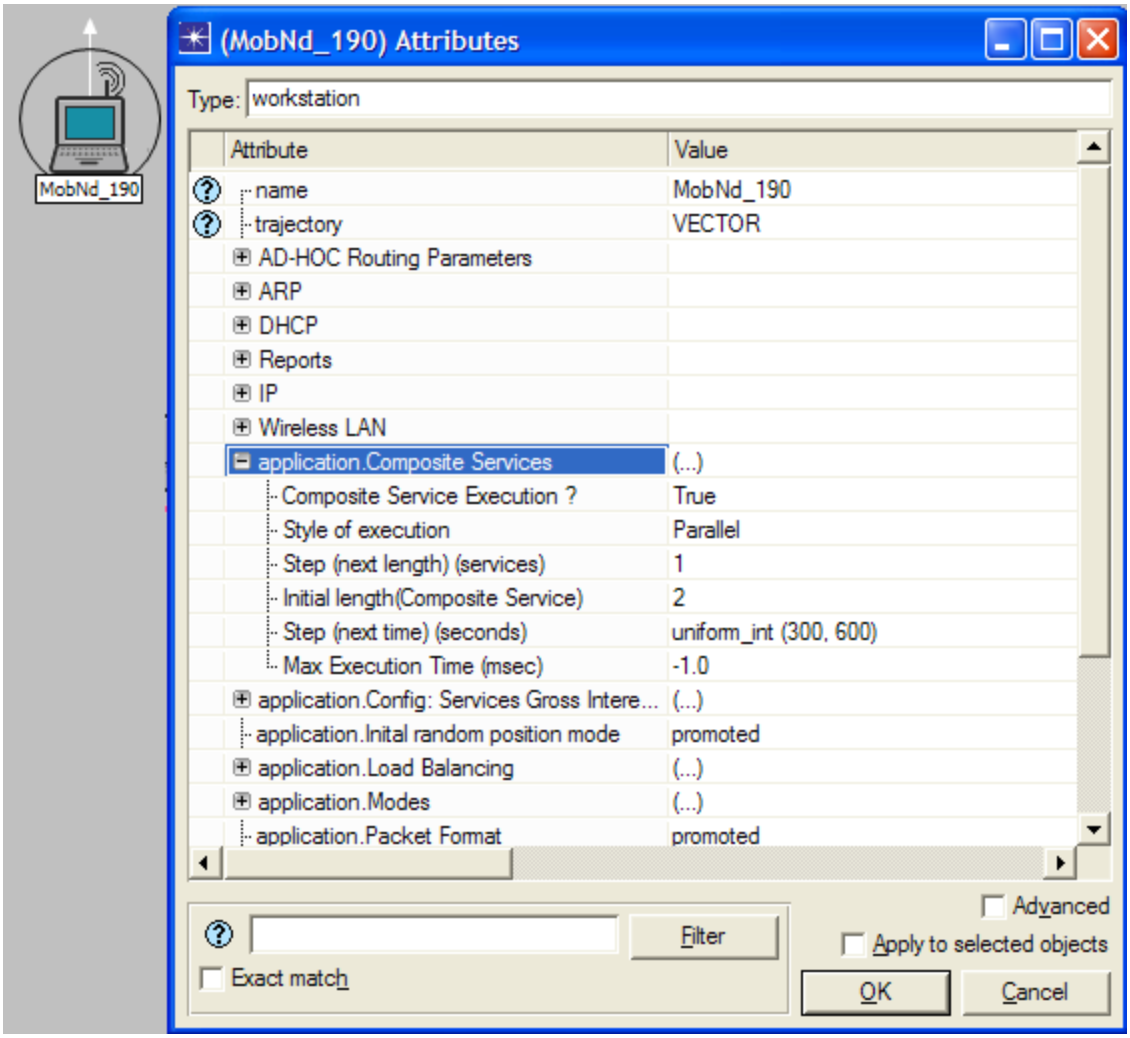

Figure 1: The mobile node fields of the process of generating the composite service requests

Selecting the required Statistics

As mentioned before, the main statistics for evaluating the SDP performance are stored in the repository model. You may select them by:

1. Right click on the “repository” node.
2. Choose “Choose individual DES Statistics”.
3. Navigate in the “Module Statistics” to “Repository”
4. Select your desired matrix from the delivered different categories of SDP statistics.

The new added statistics to indicate the performance of the new SDP-based composite service execution are contained in the tab “Composite Service”. It contains two new measurements of the composite success ratio (which reflects the ratio between the number of the succeeded composite requests of a specified length to the total number of generated composite requests

from the same length) and the composite response time (which is the average of the round trip response time of the succeeded composite requests of the same length). Both of these newly added measurements are dimensional. The first index [0] of both measurements are neglected. The index of each measurement indicates the composite service “length-1”.

Please refer to the user manual of [1] to get more details about the following subsections.

Notes on the results

Do not Forget

Part III: Programmer Guide

SDP Packets' standard

Getting a look at how the SDP application deals with the packet sending process, message structure and interrupts will be useful in understanding of the SDP packets' standard.

So, please refer to the previous programmer guide in version 1 in [1].

Message Types and Interrupts

The previous version of SDP implementation [1] introduced a set of messages structures for a different set of packets which present the exchanged information between the network nodes in any SDP-Opnet-project. The values of the field of "MsgType" identifier, which is defined and presented in [1], are extended in this documentation. Table 1 shows and describes the new message types and the involved sending and receiving parties of each of them.

MsgType	Interrupt	Sent by	Received at	Description
2	SDP_Request	Host	Host	Service query / Composite response.
3	SDP_Reponse	Host	Host	Service response / Composite response

Table 1: Different SDP message types and descriptions.

MsgType	Interrupt	Contents' list (parameters are separated by "|")
2	SDP_Request	|ClientID|SerID|RepID| Time of request||for_quality_reasons?(T/F)| |compositeRqst||CompositeRqstStructure|
3	SDP_Reponse	|ProviderID|SerID|RepID|Time of request|+BusyIndctr+|compositeRspns|+|compositeRqstStructure|

Table 2: the Contents' list structure w.r.t. the message type and interrupt.

Table 2 presents the Contents' list structure with respect to the message type or interrupt of the new or modified packet structures based on the previously mentioned contents' list structure in [1]. Note that the abbreviation used here are as followed:

SerID = service id, Name = service name, RepID = replica id, ReqSizeDisk= required size on disk for a service to be hosted (excutable and configuration files), ReqSizeMemory = required memory size for a service to be run, ReqIndex = service requirements' index value review [1], Ser1..N = Service id, and status= status of the service provider (active/iactive), and (T/F) indicates a Boolean value.

Composite Request Structure

A new structured field for managing the process of sending composite service requests and receiving composite responses has been added to be contained by the exchanged requests' and responses' message structures.

Figure 2 shows a configuration entry definition, in the delivered code : struct "Composite_request_configuration", which holds the execution information about both of composites request and response. Besides, holding the information about execution style, an extended descriptive record (indicator entry, in the delivered code: struct "list_indicators_entry") about the current status of the execution plan which is usually better to be kept on the requester side since, the delivered SDP-based execution implementation is centralized(see [3]). Once, any of the atomic-composite service responses arrived at its original requester, the related indicator entry of this request should be found based on the value of the indicator id field. If all of the composite service responses arrived (or a time out detected),the related indicator entry will be deleted at the requester side.

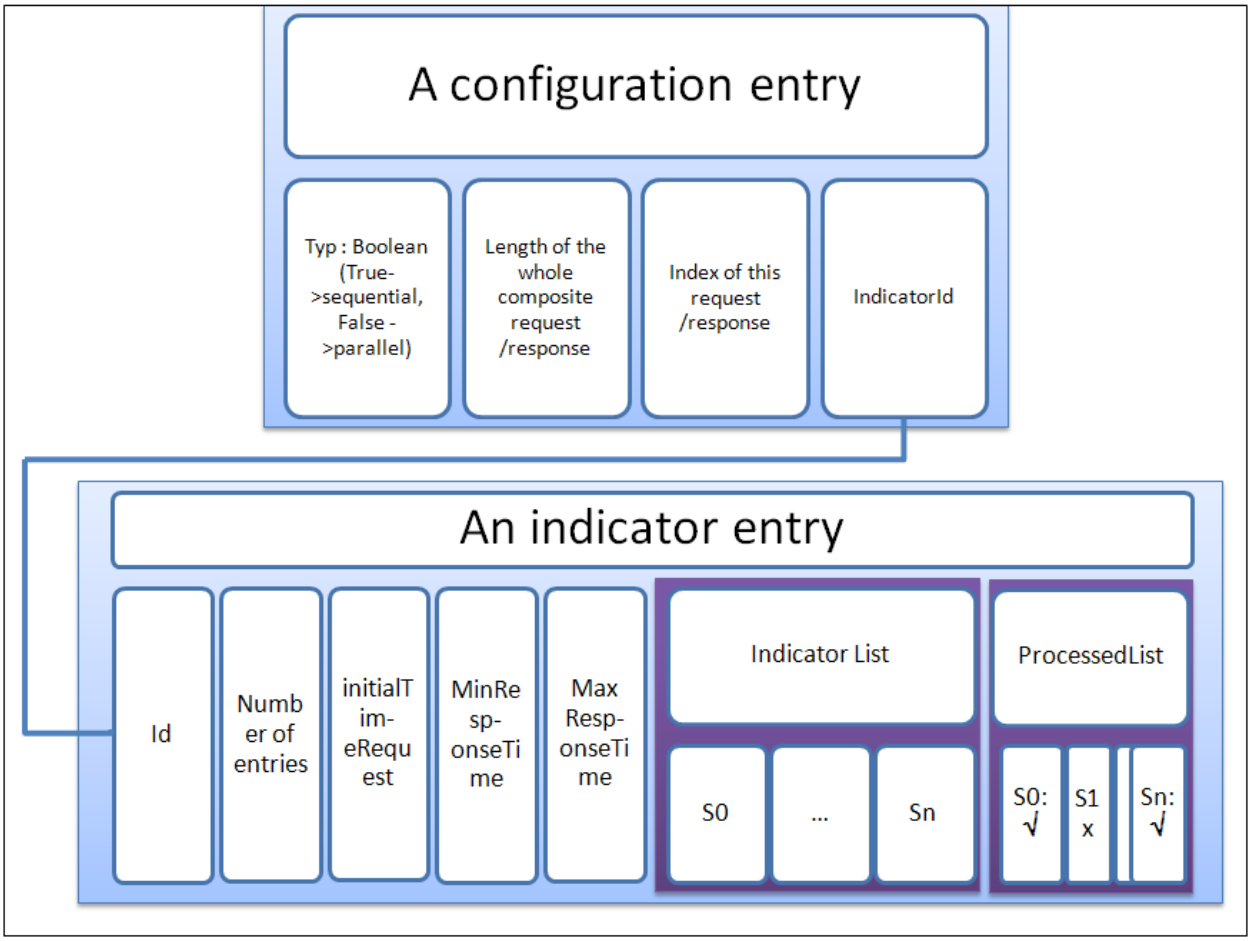

Figure 2: A configuration entry with its related indicator entry

SDP-based Composite Service Execution Messages

The current version of SDP-based execution is using the set of introduced packet formats of [1] with minor modifications for request-response message structures. The following listings 1 and 2 show the different logic behind generating and sending the messages of both sequential and parallel composite service requests.

```
for (L = 2;    L ≤ MaxNrDefinedServices;    L = L+Step)
   empty CompositeRequest_i
   for (J = 1;    J ≤ L;    J++)
      intiate(CompositeRequest_i);
      send_request(Service(J));
      wait
      until arrived_response_form(response_j,Service(J));
            proceed;
      otherwise
            break;
```

Listing 1: The logic in the sequential composite service request generation

```
for (L = 2;    L ≤ MaxNrDefinedServices;    L = L+Step)
   empty CompositeRequest_i
   for (J = 1;    J ≤ L;    J++)
      intiate(CompositeRequest_i);
      include(CompositeRequest_i,Service(J));
      send(CompositeRequest_i);
```

Listing 2: The logic in the parallel composite service request generation

The following components and process are mentioned in details in [1].

- Repository Node
- Process Model
- Repository Listener implementation
- Statistics and Readings

SDP Mobile Node

In the file "CompServExe SDP node arch.nd.m", the node architecture of a mobile node is implemented (to be easily included in any SDP project). It is the same mobile node architecture

of [1]. Please refer to the documentation of [1] to have more details about the implementation of fitting a SDP-application in the proposed node architecture.

Process model

Based on the processes model which has been represented in [1], the process model of SDP application processor is slightly modified. As shown in Figure 3, two new states have been introduced. These new states are mentioned as follows:

- CompRqst: based on the specification of the composite service requesting behavior which have been introduced in the "How to specify" subsection in the user manual, the mobile node schedules the invocations of this state. Based on the style of the composite service execution, this state calls and controls the flow of generating the requests to the atomic services which are included by a composite service request.
- TestComp: reserved to be used in further implementations.

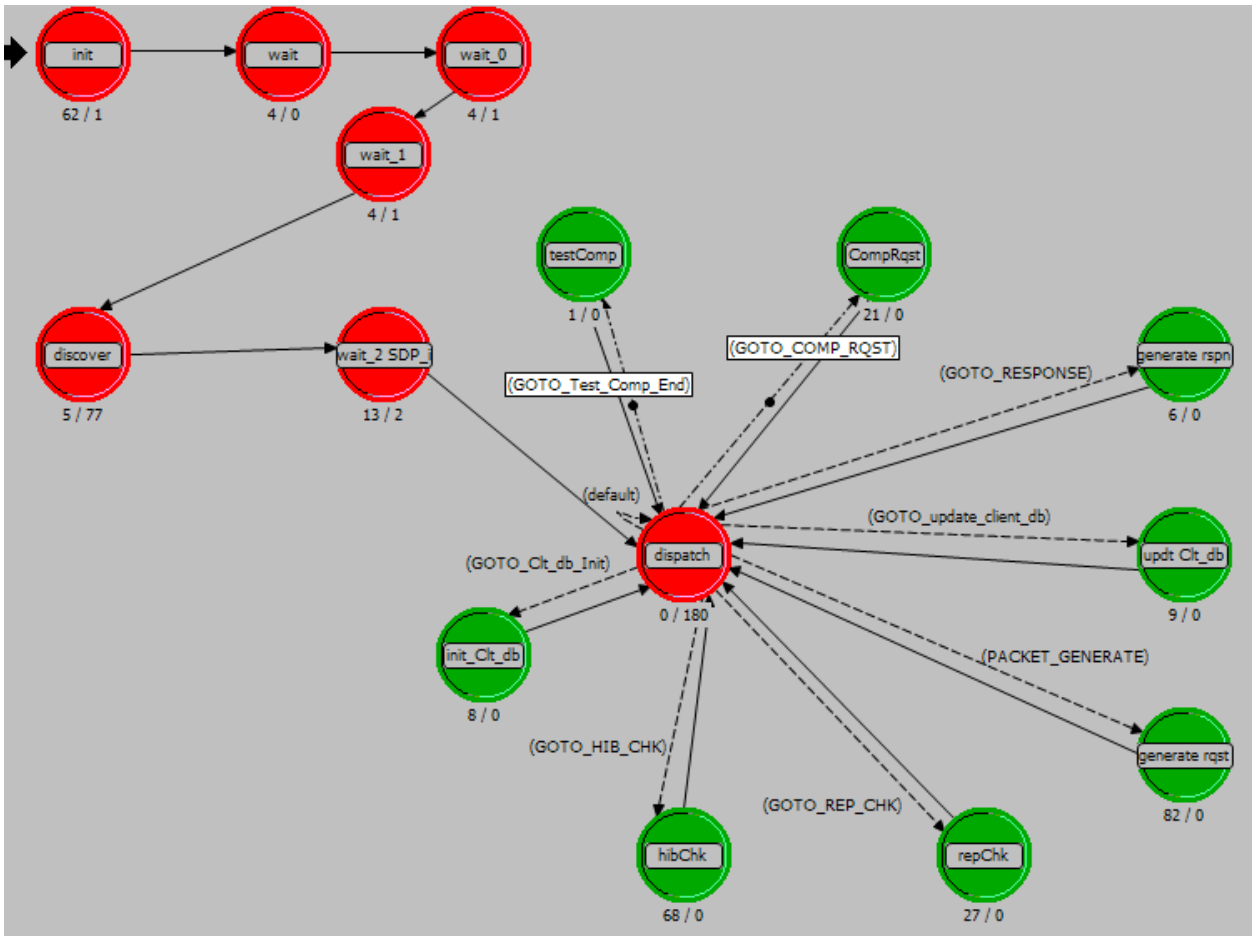

Figure 3: The new process model of a SDP application for composite service execution.

Interactions among the Mobile nodes and the Repository

Please refer to [1] for more details about the following processes and components:

- Service initiation process
- Service request-response process
- Replica forward (service replication) process
- Service hibernation process
- Service restoration process
- Neighbor wakeup process (Wakeup neighbors' mode)
- Forward inactive replica process (Forward Inactive replicas' mode)
- Load balanced service request process
- Service Selection Modes
- Pipeline stage

Composite Service Execution Styles

The composite service execution process organizes and controls the tasks of composite requests and responses for the atomic deployed services. As mentioned before, the current implementation of a SDP-based service execution aims to draw the boundaries of composite service execution based on SDP. The performance of any composite service execution will vary between two extremes of (namely: "Sequenatial" and "Parallel") execution styles. The current implementation considers a central coordination for controlling the whole execution processes where each of the clients (who issue the composite service requests) is supposed to look after its own requests for the atomic services and organize the results through the different execution steps as shown in Figure 4.

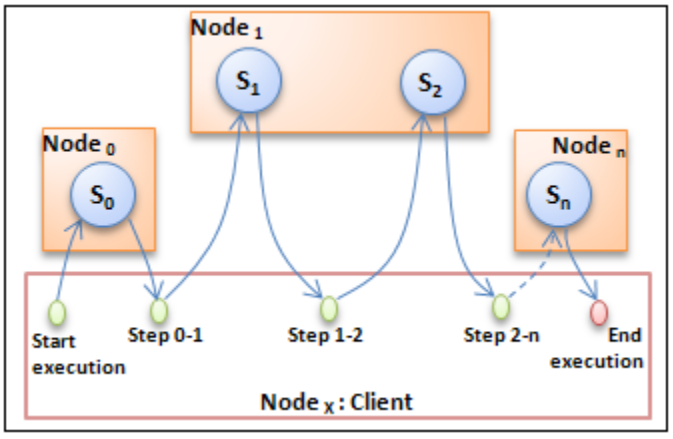

Figure 4: Centralized composite service execution

Sequential Execution Style

As shown in Figure 5-A, the client starts executing the composite service request of length n (the number of the atomic services which have been included in the request is n) by sending the first service request to the first service provider at time T=0 then, after the results of the service S0 arrived, the second request will be sent to the next service provider and so on until the end of the composite request. If there is any applied time constraint then, the execution flow should be examined during the drawn execution steps .

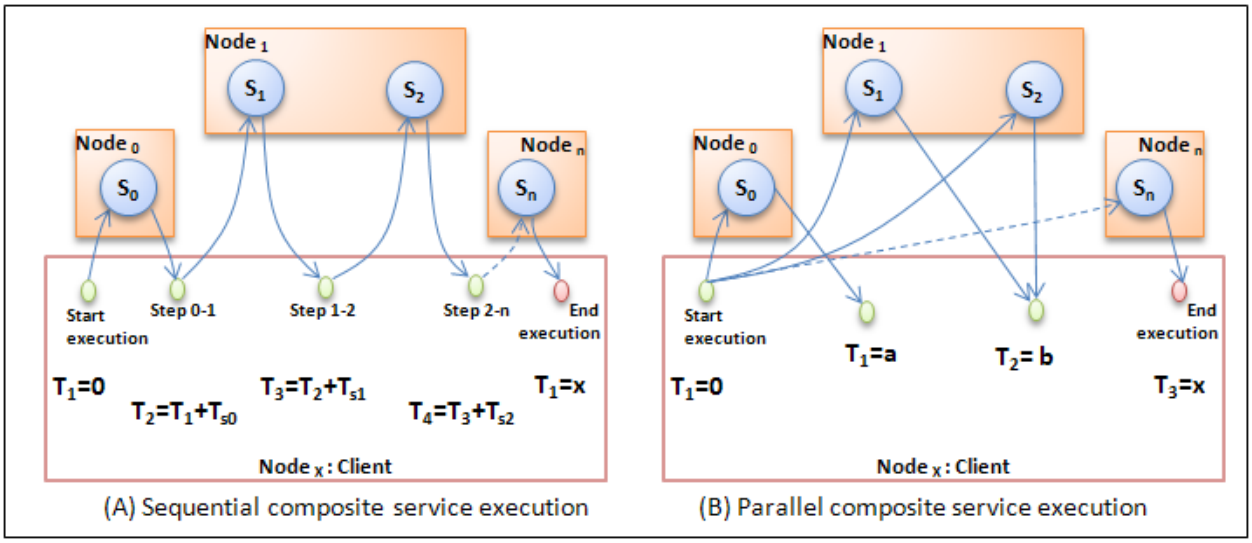

Figure 5: Sequential and parallel execution flow and control of composite service execution

Parallel Execution Style

As shown in Figure 5-B, the client starts executing the composite service request by sending to all the atomic service which have been included by the composite request at once. The execution steps here is just a set of time indications of the atomic service results' arrivals at the client.

Source code function prototype explanation and Lists-structures prototypes

This section contains only the added functions, structures and the important lists for the composite service execution. So, please refer to [1] to get more details about the rest of the source code.

Repository

1- Performance matrix functions

void compute_statisticsGrIGrIII (PrgT_List* net_partitions);

Inputs: a partition list .

Outputs: void.

Comments: computes and writes on the statistics' handlers the service distribution processes performance of SDP for all services.

2- Important structures' prototypes

struct Composite_service_statitics_entry; // To hold the information of probing the composite service execution performance matrix in the list of "Composite_service_statitics".

PrgT_List* \Composite_service_statitics;

Mobile Node

1- Composite Service Execution functions

void send_composite_service_requests_parallel();

Inputs: void

Outputs: void

Comments: manages the processes of generating and sending a set of parallel-style composite service requests to the atomic services. Each time this function is called, it iterates the composite request generation process. It start at the initial given length of the composite requests then it increments this length by a given step (length of services).

For example, if we have 5 {S1, S2, …, S5} service definitions at the repository and the initial composite request length is 2 services and step of 1 service then, once this function is called, it will generate 4 composite service requests as follows: the 1st (S1,S2), the 2nd (S1,S2,S3), the 3rd (S1,S2,S3,S4), the 4th (S1,S2,S3,S4) and the 5th (S1,S2,S3,S4,S5). So, the requester poses at the same time after this function is invoked the four previously mentioned composite request which require to be executed in parallel execution style.

---

void send_composite_service_requests_sequential();

Inputs: void.

Outputs: void.

Comments: the same as the “send_composite_service_requests_parallel” function but the generated composite services are required to be executed in parallel execution style.

Bool send_a_sequential_composite_request(int L, list_indicators_entry* indicator);

Input: length of a sequential-style composite service request, the list of the local indicators where information about this composite request will be saved.

Output: Boolean if the whole request is successfully sent.

Comments: sends an only one specified atomic request of length L to be executed in sequential style.

---

void receive_composite_service_response(Message_record* CompRspns);

Input: a message structure of an atomic service response.

Output: void.

Comments: to be called once the listener detects that the service response is belonging to a composite service response. This function receives any type (sequential/parallel) of the composite service responses.

---

void received_parallel_composites_response(Composite_request_configuration* config, double RoundTripTime);

Input: a configuration record and the related response time of getting its message.

Output: void.

Comments: for the parallel composite requests, it receives and initiates examining if all of the atomic responses of a composite response are received.

---

void sreceived_sequential_composites_response(Composite_request_configuration* config, double RoundTripTime);

Input: a configuration record and the related response time of getting its message.

Output: void.

Comments: for the sequential composite requests, it receives and initiates examining if all of the atomic responses of a composite response are received.

---

bool all_composite_responses_arrived(list_indicators_entry* indicator);

Input: the list of current not finished indicators at the current client.

Output: returns true if a specified composite service is executed (at certain time).

Comments: checks if all of the responses regarding a specified composite request is already collected.

2- Important lists' and structures' prototypes

PrgT_List* \List_of_Indicators; // Holds the minor composite requests' indicator for the composite service execution configuration record locally at the requesters.

struct Composite_request_configuration; //To be included in both of service requests and responses message structures.

struct list_indicators_entry;// Represents the indicator record to be saved at the "List_of_Indicators" list.

References

[1] Mohamed Hamdy and Birgitta König-Ries. SDP-Implementation in Opnet® v.1: MANETs' Service Replication and Load Balancing -Release Notes-User Manual–Programmer Guide, https://enterprise1.opnet.com/tsts/4dcgi/MODELS_FullDescription?ModelID=944, FRIEDRICH-SCHILLER-UNIVERSITY OF JENA, 2010.

[2]Mohamed Hamdy and Birgitta König-Ries. Book of Communications in Computer and Information Science, Book of the selected papers of the ICETE 2008, volume 48 of CCIS 48, chapter: The Service Distribution Protocol for MANETs- Criteria and Performance Analysis, pages 467-479. Springer Berlin Heidelberg, 2009.

[3] Chen et al, "A Dynamic Execution Path Selection Approach for Composite Services in MANETs," The 4th International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM 08), Chengdu, China, 2008.