Requirements-Aware, Template-Based Protocol Graphs for Service-Oriented Network Architectures

Thesis approved by

the Department of Computer Science of the University of Kaiserslautern

for the award of the Doctoral Degree

Doctor of Engineering (Dr.-Ing.)

to

Abbas Siddiqui

Date of Defense:	27.09.2016
Dean:	Prof. Dr. Klaus Schneider
Reviewer:	Prof. Dr. Paul Müller
Reviewer:	Prof. Dr.-Ing. habil. Andreas Mitschele-Thiel

D386

This work is dedicated to my parents - *Mr. Noor Ali Siddiqui* (in loving memory) & *Mrs. Siddiqui* - for their unconditional love and endless support, and to my brother - *Dr. Atif Ali Siddiqui* (M.D.) - for his encouragement on every path of my life

Acknowledgments

The ideas presented in this thesis are developed in the G-Lab project [1] funded by BMBF under the supervision of Prof. Dr. Paul Müller. Foremost, I would like to thank him for giving me the opportunity to work on the project. I would further like to express my gratitude to Prof. Dr.-Ing. habil. Andreas Mitschele-Thiel for being the second supervisor for the work. I am particularly very grateful to Dr. Bernd Reuther for being my advisor. He was very open towards my presented ideas. Besides, his valuable comments, suggestions, and feedback always gave me new prospects on my work. I would also like to thank my colleague Denis Schwerdel for his practical insightful on my work. Inputs of Rahamatullah Khondoker, Nathan Kerr, and Daniel Günther also played a role in ideas development. I am also thankful for the positive critics of my colleague Joachim Götze. I want to thank Mrs. Hahn and Mrs. Younis for all the administrative work and their kind nature. I would also like to thank all of my project partners for their discussions during the meetings. Last but not least, I would like to thank my family and my wife "Patricia Siddiqui" for their constant support and patience.

[1] http://www.german-lab.de/

Abstract

Rigidness of the Internet causes its architectural design issues such as interdependencies among the layers, no cross-layer information exchange, and applications dependency on the underlying protocols implementation.

G-Lab[2] is a research project for Future Internet Architecture (FIA), which focuses on problems of the Internet such as rigidness, mobility, and addressing. Where the focus of ICSY[3] was on providing the flexibility [Mue13] in future network architectures. An approach so-called Service Oriented Network Architecture (SONATE) [KSMB14]) is proposed to compose the protocols dynamically. SONATE is based on principles of the service-oriented architecture (SOA) [Erl08], where protocols are decomposed in software modules and later they are put together on demand to provide the desired service.

This composition of functionalities can be performed at various time-epochs (e.g., run-time, design-time, deployment-time). However, these epochs have trade-off in terms of the time-complexity (i.e., required setup time) [Bas80] and the provided flexibility. The design-time is the least time critical in comparison to other time phases, which makes it possible to utilize human-analytical capability. However, the design-time lacks the real-time knowledge of requirements and network conditions, what results in inflexible protocol graphs, and they cannot be changed at later stages on changing requirements. Contrary to the design-time, the run-time is most time critical where an application is waiting for a connection to be established, but at the same time it has maximum information to generate a protocol graph suitable to the given requirements.

Considering limitations above of different time-phases, in this thesis, a novel intermediate functional composition approach (i.e., Template-Based Composition) has been presented to generate requirements aware protocol graphs. The template-based composition splits the composition process across different time-phases to exploit the less time critical nature and human-analytical availability of the design-time, ability to instantaneously deploy new functionalities of the deployment time and maximum information availability of the run-time. The approach is successfully implemented (i.e.,A.6), demonstrated (i.e., [GSS+12]) and evaluated (i.e.,6) based on its performance to know the implications for the practical use.

[2]http://www.german-lab.de/
[3]www.icsy

Contents

List of Figures

Chapter 1

Introduction

On the Internet, the network stack is divided into distinct layers that can be implemented by different protocols. Each layer offers a service to directly adjacent layers. This crisp and robust design has provided its advantages like functionality scoping and stability. However, it also has disadvantages like as following:

- Protocols on different layers implement the same functionality like IP and TCP Checksum.
- Architecture does not support the cross-layer optimization for instance the physical layer can not optimize the error-correction or data-coding (e.g., multimedia over wireless) to the application requirements.

The protocol stack, we use today, has been introduced decades ago. Since then the whole Internet came to existence and with it the single protocol stack: TCP/IP. What was a good solution back then, is no longer appropriate to fulfill emerging demands of applications. As technology advances, new requirements arise regarding security, QoS, mobility support (e.g., Mobile IP), privacy, sustainability and scalability. However, development of the Internet during the last years has shown that it is difficult to integrate new functionalities [Han06]. Especially the core mechanisms (TCP/IP) are hard to change. It is not a problem of a single protocol or its implementation it is rather related to architectural limitations that have been around since the commencement of the Internet.

The following are some issues that cause the rigidness of the Internet architecture.

- Introduction and Exclusion of a Functionality: Including/excluding functionality in/from a network stack or replacing an entire stack requires a complicated deployment and migration phase. In addition, the cost increases if modification of

software & hardware is needed. As a result, new functionalities are deployed at the application layer, some of those examples are security, mobility, quality of service.

- Layer Proliferation: Driven by the demands of ever emerging applications and the capabilities of new communication networks, many workarounds have been introduced like sub-layer proliferation and middle-boxes (e.g., firewalls) that were not part of the original design. MPLS [RVC01] has been introduced to support traffic categorization so as time optimization. Virtual networks are introduced at layer 2.5 and, the IPsec [FK11] is introduced for securing the valuable data at layer 3.5.

  In addition to the violation of design principal, the induction of sub-layers causes the more severe problem of unknown dependencies among the layers. Modification of data by a layer can hinder the underlying layers to function properly such as IPsec encrypts the data of above layers (e.g., Transport Layer) and prohibits the proper use of the port numbers or causes the other known problem with NAT [AD04]. These extra layers also decrease the Maximum Transmission Unit (MTU). As a result, packets might be dropped or rejected at some routers.

- Middle-boxes: Another workaround is the use of middle-boxes like NATS [EF94], firewalls, proxies and gateways, which erode the end-to-end model. It also causes that edges are oblivious to newly deployed functionalities of network. As a result, middle-boxes interfere with the traffic and restrict some functionalities to work properly. It is also a hurdle in deploying new protocols in a network as unknown traffic is discarded or blocked by some middle-boxes.

- No Cross-layer Communication: Because of the lack of cross-layer communication in the current architecture, different layers implement the same mechanisms such as IP and TCP Checksum. It influences the data transmission rate and wastes the valuable computing resources. In the worst case, the execution of particular mechanisms can be counterproductive such as TCP congestion control in wireless networks, and there is no way to inform a layer to disable the certain mechanisms.

The presented work is carried out under the umbrella of the G-Lab[1] project sponsored by the Federal Ministry of Education and Research (BMBF), which focuses on "Future Internet Architecture" It investigage many aspects of the architecture such as addressing,

[1] http://www.german-lab.de/

mobility, cross-layer composition, security, and routing. Our group, Integrated Communication Systems (ICSY), at University of Kaiserslautern took part in research on the flexibility of the network architectures and cross-layer composition.

The rest of the chapter describes the motivation, followed by the problem statement, the methodology, and the structure of the entire thesis.

1.1 Motivation

The inflexibility of the Internet is a motivation to combine various functionality to achieve the desired communication service. The design of the Internet has lead to a rigid system that has limited abilities to keep up with the constantly changing demands of applications like increasing data rates, reliability, QoS, network heterogeneity, mobility, and security. The following is the discussion on the desired flexibility in Future Internet Architecture (FIA) and on-going research on the topic. The problem for the proposed approach is described in section 1.2 .

The FIA research[2] [SZ09][3] [Pee11][4] divides into two distinct views so called "Evolutionary Design [RD10]" and "Clean-Slate Design [Fel07]". The Functional Composition (FC) [SLee11] is a clean-slate design approach. The idea of FC is borrowed from service oriented architecture [Erl08] [RSSM09], FC considers the Internet as a distributed software system. Many projects [BFH03, TWP06, DRB+07, MR08] have used the FC approach. The state-of-the-art of the approach is presented in [HSee10] and challenges in [SKR+11]

The FC is an approach to achieve flexibility that is one of the major goals [MDAD] of FIA so that unseen futuristic demands can be accommodated without having a need to change the design. The flexibility [Mue13] can be divided into two sub-goals so-called long-term and short-term flexibility.

- Short-Term Flexibility: It describes adaptability of a network for the given conditions, requirements and constraints. A network should be adaptable to user demands and yet it takes into account the given network conditions (e.g., bandwidth, jitter, delay) besides the provided policies defined by such as a domain, a service provider, and an administrator.

[2]Global environment for network innovations (geni) http://www.geni.net/
[3]http://www.german-lab.de/
[4]http://www.fi-ppp.eu/

- Long-Term Flexibility: It characterizes the evolution of networks with ongoing technological development. It entails aspects such as inclusion, exclusion or exchange (i.e., exclusion + inclusion) of the given functionalities besides, updating the existing ones.

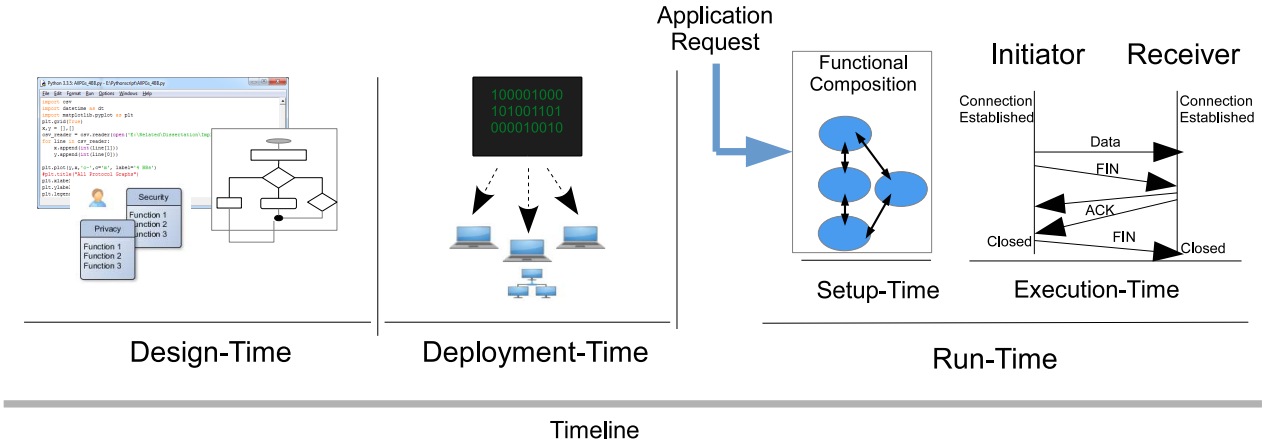


Figure 1.1: Time Phases

One of the key differences among the functional composition approaches is time phase they are performed at. The fig. 1.1 shows the three time-phases, the *run-time* begins once application/user sends a connection request to an underlying network. The run-time is further divided into the setup-time (i.e., the time required to compose a protocol graph) and the execution-time (i.e., is a time when actual data transmission starts till the connection closes). This classification of the run-time is needed as an adaptation (i.e., change of a PG is based on variation in conditions such as of network) of protocol graphs is taken place at the execution-time, while a PG is composed at the setup-time.

The *design-time* is used for protocol design & implementation phases, and the designed protocols are deployed to systems at the *deployment-time*. **Time phases differ regarding available information**; the design-time has assumed applications requirements and, no information about host environment or network conditions. While the run-time has actual knowledge of application requirements, host environment and network conditions. Whereas the deployment time has a slight edge on the design-time by having the host information, unless a service is developed exactly at the system where it is deployed.

1.2 Problem Statement

The composition at the design or the deployment time phase is less time critical than a composition at the run-time as no user or application is waiting for a connection to be established but at the run-time, it is otherwise. However, if a composition is performed at the design or the deployment time, it cannot take into account the information that is available at the run-time that restricts its ability to be optimized.

The run-time composition gives an ideal possibility of optimizing by combing functionalities as late as possible so that a composition process has maximum information availability to achieve an optimal composition. As a result, it also increases the required setup-time as it has to perform all of its tasks at the setup-time.

It is more likely that a run-time composition uses an automatic algorithm. Any human-involvement will delay the process further. Higher information availability at the run-time increases the number of inputs for a composition method, hence the required time for a successful composition.

The following are the factors, which increase the required time in a composition approach.

- Placement: To place functionalities in proper order is one of the vital tasks to carry out in a successful composition of a PG.

  Dependencies among functionalities and optimization of a service are the reasons to have a proper placement of functionality in a protocol graph. In the conventional network stacks such as TCP/IP, every functionality has a fixed place which is carefully chosen by experts. On the one hand, this has an advantage of having an optimal order of functionalities in a service, but on the contrary, because of its static nature, the introduction of new functionalities or exclusion of existing ones is very hard.

  Dependencies among functionalities can be observed in the TCP such as in ordering of acknowledgment and sequence number. The acknowledgment mechanism can not work until sequence number is not known as this number is needed to set the next expected sequence number.

  The given examples in fig. 1.2 can further elaborate the importance of placement; wherein fig. 1.2(a), packetization is placed after re-transmission, in this case re-transmission will not work as it requires data in packets form to stamp (identify), to count and to retransmit. Even though a protocol graph may cover the desired func-

tionalities but because of wrong placement, it will not be able to execute properly. If an end-to-end encryption mechanism is placed after an addressing mechanism (i.e., as shown in Fig. 1.2(b)), the address will be encrypted and routers will not able to forward the incoming packets to the destination or a compression after an encryption is not as effective.

Placement of functionalities is effected by the explicit requirements such as from application, network, and domain. Moreover, the implicit requirements of optimization such as order between compression and encryption mechanisms.

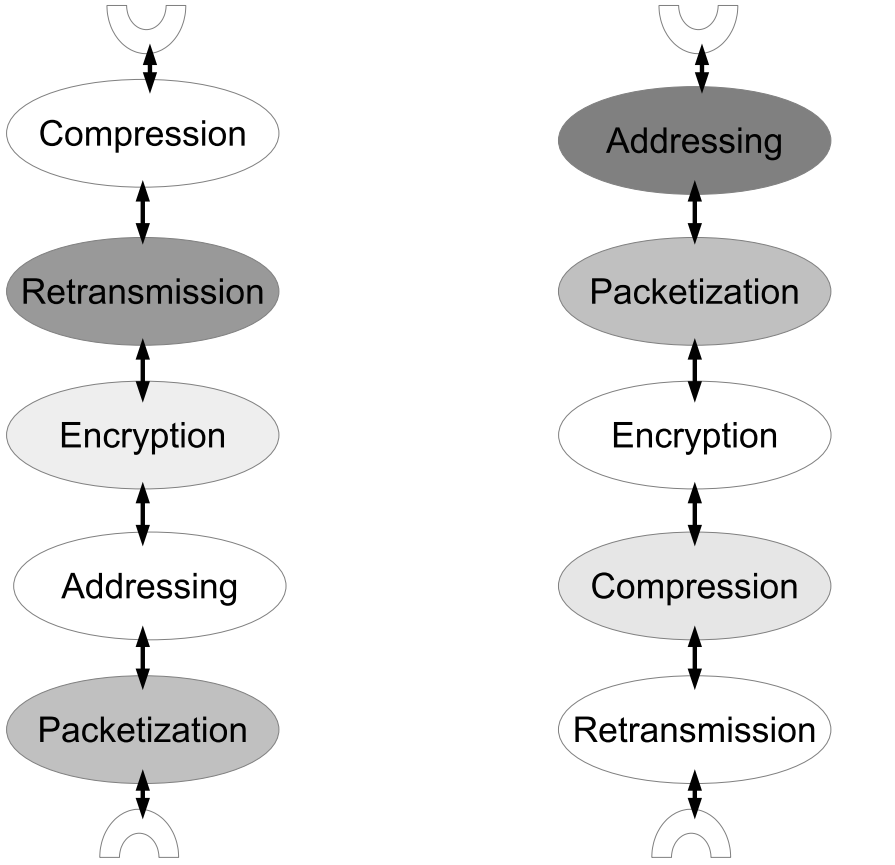


Figure 1.2: Examples of Placement

Granularity is a major factor in the placement issue as it increases dependencies among functionalities and ultimately the need for proper placement. Such as error-correction and error-detection can be a single mechanism or separate mechanisms. Error-detection-correction (i.e., error-control) as a single mechanism eliminates the

issue of placement or dependency within each other but limits the ability to be used individually. The cases where error-correction is not required and packets are discarded once an error found can longer be realized. On the one hand, fine-grained functionalities increase those dependencies among each other but, on the contrary, it also escalates re-usability of functionality.

By using the human-analytical ability, it is easier for an expert to place functionalities in a proper order, but an algorithm requires additional knowledge (e.g., ontology) to create an executable PG and it will have an impact on the required time. A trivial composition approach will try to order all the available implementations of the requested functionalities and relies on permutations (i.e., where order matters). However, this approach may use a constraint on the size (i.e., fixed number of elements) of set that will result in $n_1 * n_2 * n_3 * .... * n_r$ or it can be written as an exponent of r: n^r. Where n is total number of building blocks and r of them are chosen for a single permutation. Moreover, if different size (i.e., r) of sets are possible then formula will result in summation of all possible permutations such as when r is from 1 to 3 the number of permutations will be: $n_1 + n_1 * n_2 + n_1 * n_2 * n_3$, this summation can be described in general way by this formula: $\sum_{r=1}^{r=n} n^r$. Where lower limit "1" restricts to have at least single BB selected and "n" is a upper limit.

- Connection of Functionalities: The incoming requirements do not provide any idea about how functionalities will be connected to each other. In a sequential PG, functionalities are executed one after other. However, if functionalities are supposed to be connected in parallel graph such as connection management and data-transmission then additional information is required to connect them. Besides, the wrong connections among these functionalities can create a loop. The composition process must check for the cyclic connections to avoid the endless loops. All those extra information and consistency checking will increase the time required for composing a PG.

- Information Availability: This can be best described by early or late binding of the communication services, the early binding will not benefit from the information such as network capacity, hardware and software constraints thus, the composed protocol graph might not be the optimal one. On the other hand, late-binding has the advantage to get real time information. However, the increment in information results in more inputs to compute which also increases overall requited setup time.

If a PG is composed at the design-time, it requires no additional composition to provide the requested service. Hence, it reduces the required setup-time. However, it cannot be further optimized for the dynamic requirements and constraints. On the contrary, all those issues must be taken care if a PG is composed from the scratch at the run-time. It can take a substantial amount of time (e.g., few microseconds to few minutes) but, the composed PG can be well adapted to various demands.

According to [BY97] "A complex system is a system formed out of many components whose behavior is emergent, which is, the behavior of the system cannot be simply inferred from the behavior of its components." By that definition, a service generated from interaction of various modules is a complex system. The algorithm complexity is a direct result of solving a complex system, an intuitive measure of the algorithm complexity will depend on amount of components, their interactions and possible combinations, besides amount of input parameters. As described above in a trivial composition approach, the algorithm complexity will accounts towards an exponential rise in the required time.

In [Sip96] author defines time-complexity as "The time complexity of an algorithm quantifies the amount of time taken by an algorithm to run as a function of the length of the string representing the input." Hence, in this work, the required setup-time and the time complexity terms are used interchangeably.

Given that generated service is a complex system, and it remains same either it is built at the design-time or at the run-time. However at the design-time, first of all, human-analytical abilities can be used, and second, time constraints are far more relaxed unlike at the run-time. However, the design-time has least available information to generate an optimal service for the arbitrary requirements.

Now the question is here, how long it will take to make the desired service available to a user once a service is requested while using maximum available information (i.e., to achieve the desired flexibility 1.1) for a composition process? The design-time fails to satisfy the requirements of using the maximum available information for a composition process, whereas, use of the run-time will increase the required-setup time significantly as argued above.

Individually, neither the run-time nor the design-time can provide an optimal solution to the problem.

This research is based on the hypothesis that the required setup-time can be significantly reduced by splitting a composition in different time-phases without compro-

mising on the desired flexibility of the FIA.

This work proposes an intermediary FC approach (i.e., template based composition), which splits a composition process across the different time phases. The interdependencies within a protocol graph are solved at the less time critical time-phase (Design-Time). The critical time-phase (Run-Time) is utilized for optimizing the composition by use of the maximum available information. The detailed approach can be found in Chapter 5.

Methodology: The research uses the classical hypothesis experimental method to address the above question.

Before proposing the approach, to have a sufficient background and to avoid the repetition of work, a literature review has been carried out which is found in the chapter 4. A prototype as proof-of-concept is implemented (code: A.6) to evaluate the performance, in terms of required time, of the proposed approach. The performance evaluation of the approach is presented in 6. The functionalities of the TCP/IP stack are used for the testing purpose and the extracted functionalities are published in [SDS+09]. The prototype of the approach has been demonstrated in [GSS+12]. In addition, the flexibility of the approach is discussed in section 5.2.

1.3 Document Structure

After this introduction, the rest of the thesis is organized as following:

Chapter 2 describes main terminology and the concept of functional composition and its requirements identified within the G-Lab project and used in this thesis. Chapter 3 covers the proposed SOA-Based Network Architecture by ICSY[5] and its components like requirements, API, composition & selection of PG and finally the execution framework (SONATE Framework).

Chapter 4 describes the state of the art on the topic of functional composition, the approaches are classified by time-phases (e.g., design-time, run-time and intermediate). Also, it covers the comparison among intermediate approaches including the template-based composition.

The proposed approach (Template-Based Composition) is described in chapter 5, at first it discusses the basic concept then next it elaborates how the templates and domain policies are described, followed by the template selection and the generation of executable protocol graphs. Later, it describes a TCP-like protocol graph generated by the template-based composition followed by the concluding remarks on the proposed approach.

The performance of the approach (i.e., based on Java-implementationA.6) is evaluated in the chapter 6. The thesis is concluded with the chapter 7 besides the future work which can be carried out in later projects. The appendices contain the examples used throughout the work, and brief detail about the implemented prototype.

[5] www.icsy.de

Chapter 2

Background

This chapter covers the knowledge that is essential to understand the research work presented here. The first section describes the terminology used in this work, the next one, at first describes the functional composition from general perspective and then go on to describe the definition and the requirements for the Service Oriented Network Architecture (SONATE) 3.2. Also, the issue of heterogeneity of services is described briefly, it is not part of either SONATE or the research in this thesis. However, it is needed to understand how functional composition can increase the diversity of services so the challenge to deal with it.

2.1 Terminology

Same terms may have different meanings in different research fields. To avoid the ambiguity of terms, they are clearly described in the individual research work, unless the definitions of the terms are well known and standardized. This section describes the most commonly used terms in this work, the other specific terms are explained within its context as needed.

According to IEEE Standard [IEE00], *architecture* defines the fundamental organization of a system, the relationship of components and design & evolution principles. The **Internet** is considered as a distributed software system [SDS+09], where several software modules (i.e., protocols/micro-protocols [GKS03]) are deployed at the end- and intermediate- nodes. The components, involved in creating a successful communication between two parties, are considered as a part of inter-network architecture such as application programmable interface (API), composition methods, selection methods and execution framework.

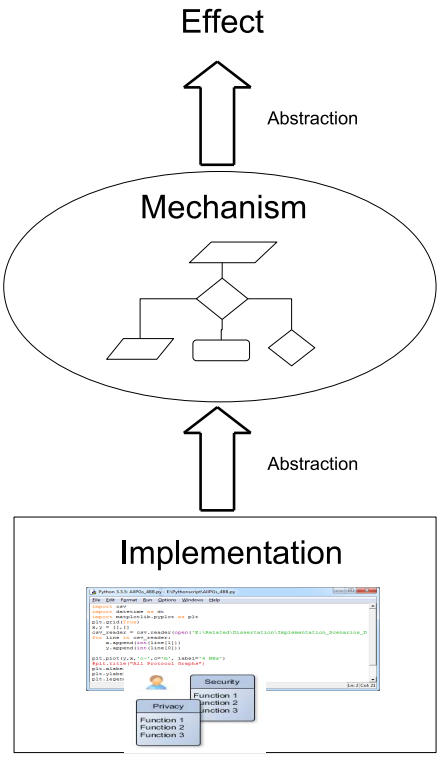


Figure 2.1: Effect, Mechanism and Implementation

In service-oriented network architecture, the idea is not to have specifically made protocol stacks anymore such as TCPIP, UDP, SCTP but rather have a functionality implemented by smaller software modules. In this work, the term **"Building block"** is used for an *implementation* of those micro-protocols.

The lowest abstraction in the fig.2.1 [SDS+09] is an **implementation**, which is a software code for a particular mechanism also referred as building block here. One higher abstraction is an algorithm or a **Mechanism** as used in this work. A mechanism provides the description of how to achieve the desired effect. Different pieces of software can implement a particular mechanism. Two mechanisms are considered equivalent if they provide the same effects regardless of their implementation techniques and code intricacy. In a network architecture, mechanisms are usually specification of protocols or micro-protocols. Such as TCP specification describes every detail which is necessary for two peers to communicate using the TCP. However, it does not define a specific implementation. There exist various implementations that are all compatible as they all implement the same mechanism. Mechanisms exist at any granularity; a mechanism can as tiny as Time-to-Live (TTL) mechanism such mechanisms are used for creating a compound mechanism.

The highest abstraction is called **Effect** as shown in the fig.2.1. An effect describes the desired outcome when a mechanism is used. An example of an effect is "security", which can partially be covered by an encryption or an authentication or any other protection mechanism. There may be several different mechanisms, which produce the same effect. For example one mechanism uses sequence numbers to keep order. Another mechanism uses a sequential acknowledgment and thereby provides the ordering as an implicit effect. In general, an effect is independent of a mechanism and its implementation, the same set of effects can be provided by a different mechanism or set of mechanisms. The correlation and examples of effects are described in detail in this paper [SDS+09].

The goal of a flexible inter-network architecture is to satisfy the customized user/application requirements, to achieve that goal the provided effects of building blocks are combined together (i.e., *composition*) to get a **Communication Service**. A communication service is an end result of a composition process. This service is later used by applications/users to establish an end-to-end connection. The terms service and communication service are used interchangeably in this text. The connections among building blocks are defined in a structured way so called **Protocol Graph** (PG) [WOP92]. A protocol graph determines which building blocks are used and how are they connected to each other in order to provide a communication service. A **Composition Method** generates a protocol graph. Composition method/process and functional composition are used interchangeably in this work.

2.2 Functional Composition

The idea of functional composition is not new it has been used since few decades in software engineering to divide a code into various modules [May72] and combine them to achieve the desired result. However, the idea has been introduced relatively late at network protocols level, as group of network researchers in early 1990s concentrated on working in dynamic micro-protocol composition. They decomposed a functionality of the protocol stacks into a set of micro-protocols and then composed those micro- protocols dynamically based on the incoming requests from an application. Some of those works are Dynamic Configuration of Protocols [VPPW93a] and Function Based Communication Subsystem [Sti94].

In SONATE, the functional composition receives different inputs from various sources such as requirements from an application, policies from a network administrator, constraints from the network and available BBs from the host system to generate protocol

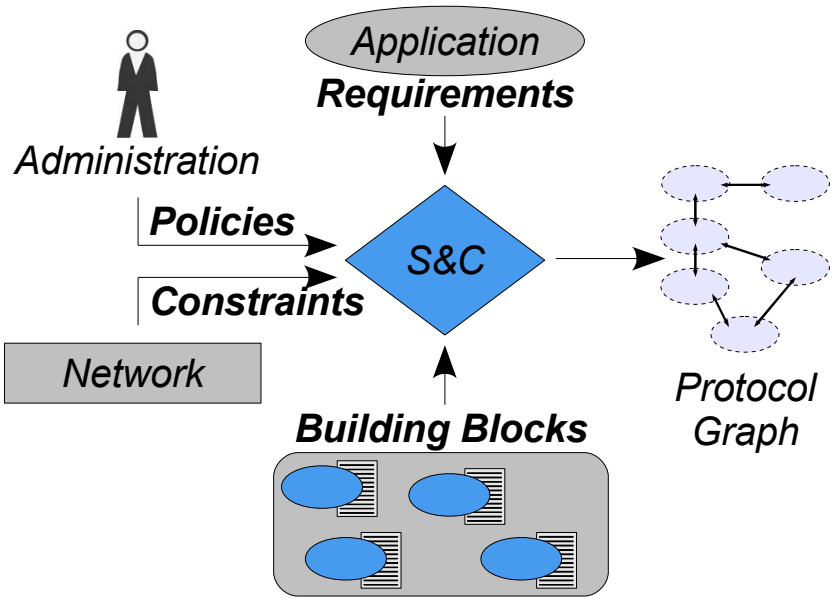


Figure 2.2: Functional Composition [KSRM12]

graphs as shown in figure 2.2.

As the topic of this work is focused on an intermediate composition, it is necessary to have an overview of the requirements of the functional composition.

2.3 Functional Composition Requirements

Certain requirements must be taken into account to develop a composition approach. In the course of the project (G-Lab), some members of the G-Lab team worked to identify the following requirements for a successful composition approach it was also published in a paper [SKR+11].

2.3.1 Description of Requirements

Applications need to be able to state the requirements to an underlying network to provide a customized functional composition. Application requirements must be described in a way that can be processed by a composition method. It is required to have a common description language and domain understanding that will be shared between application layer and a composition process. A common description language is necessary so that an application can make a request of effects that should be processed and provided by a network. There is not only a need for a description language that tells how to write

down requirements but also what to write down. Additional requirements may arise depending on an application or the platform on which application is running. The same application might have different energy consumption needs on a desktop or a mobile phone. User requirements might involve control over costs caused by consuming a certain service. An application can describe its requirements regarding optional and mandatory service properties (e.g., max delay, min data rate, max jitter). It can be indicated to the underlying network which requirements can be left out in case they cannot be fulfilled. An application can also restrict quantitative properties by providing some mini-ma and maxi-ma of the property it can be regarded as constraints from an application side.

2.3.2 Description of Building Block

A Building Block (BB), as shown in figure 2.3, is an elementary component of functional composition approach which poses various challenges. As mentioned earlier, a building block contains an implementation of a communication service or a partial communication service. A BB comprises of interfaces that enable the interaction with other entities. To define these interfaces is challenging because they must be as generic as possible so that it is not required to alter or re-implement interfaces if it needs to interact with another building block(s). A semantic description of BBs is required to achieve a functional composition. A building block's description consists of information about interfaces related to the particular mechanism.

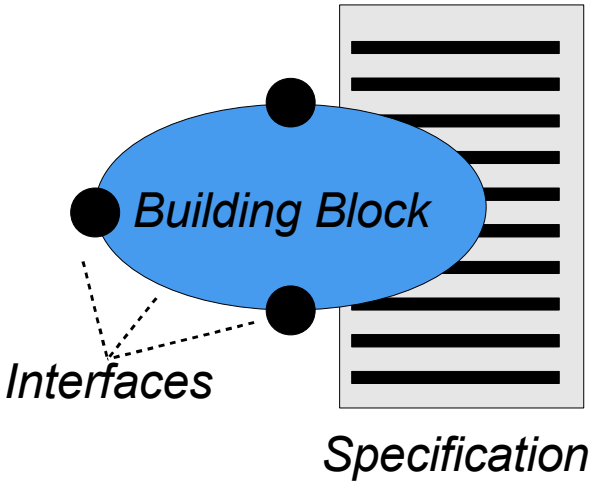


Figure 2.3: Building Block Description [SKR+11]

To describe placement (i.e., ordering) dependencies of Building blocks is another challenge but of course it depends on how services have been realized. If a single service has

been built up by single or multiple BBs.

2.3.3 Identifying the Dependencies

One of the key requirements of network functional composition approach is to identify dependencies among mechanisms.

If it is assumed that a communication service is implemented by several BBs, then it is required to identify the BBs that are involved in the implementation of a specific service and their dependencies before the composition process begins. An example of dependency can be compression and encryption mechanisms it does not make much sense to compress encrypted data. The task of a functional composition process is to create a protocol graph. An execution sequence of mechanisms can only be established if it is known that how they are dependent on each other. Dependency information is required to decide if it makes sense to have two similar kinds of mechanisms in the same protocol graph. An example of such a dependency can be call-forwarding and answering-machine mechanisms. If both of those mechanisms are included in a communication service, it will always be unclear which mechanism should be used when a call arrive. In case if a default mechanism has been specified then, another mechanism will never be executed thus it will not have any use. The mechanisms can be dependent based on their read/write, input/output relationship or their dependence can base on their optimal functioning. These dependencies might be described statically or can be obtained dynamically by observing their read/write, input/output, syntactic and semantic information. To conclude, identifying and describing dependencies and obtaining dependency information during run-time is a challenge and needs to be tackled before and/or during the composition process.

2.3.4 Finding Granularity of Mechanisms

It is a challenge to find an appropriate granularity of mechanisms that can be used by the functional composition approaches because the performance (i.e., regarding delay, cost) of different approach has various amount of impact on granularity. Granularity has also an impact on how mechanism should be described. For example, a mechanism can be an entire communication service such as ”reliable transmission” or it can be further divided into a set of mechanisms: data correctness, completeness and order preservation to make fine-grained services [SDS+09]. On the one hand, such kind of fine granularity makes functional composition flexible and on the other hand it requires a complicated description

of protocol graph(s) and building block(s) which also makes composition process time consuming. On the contrary, coarse-grained (e.g., Printing mechanism, Naming and Addressing) mechanisms are less complicated to describe but make an approach less flexible as micro-mechanisms can be not be combined with other mechanisms to create a different communication service. Nevertheless, it simplifies a composition process in terms of the number of mechanisms and their description handling.

2.3.5 Composition Methods

Finding and selecting a suitable composition method based on domain requirements is a challenging task.

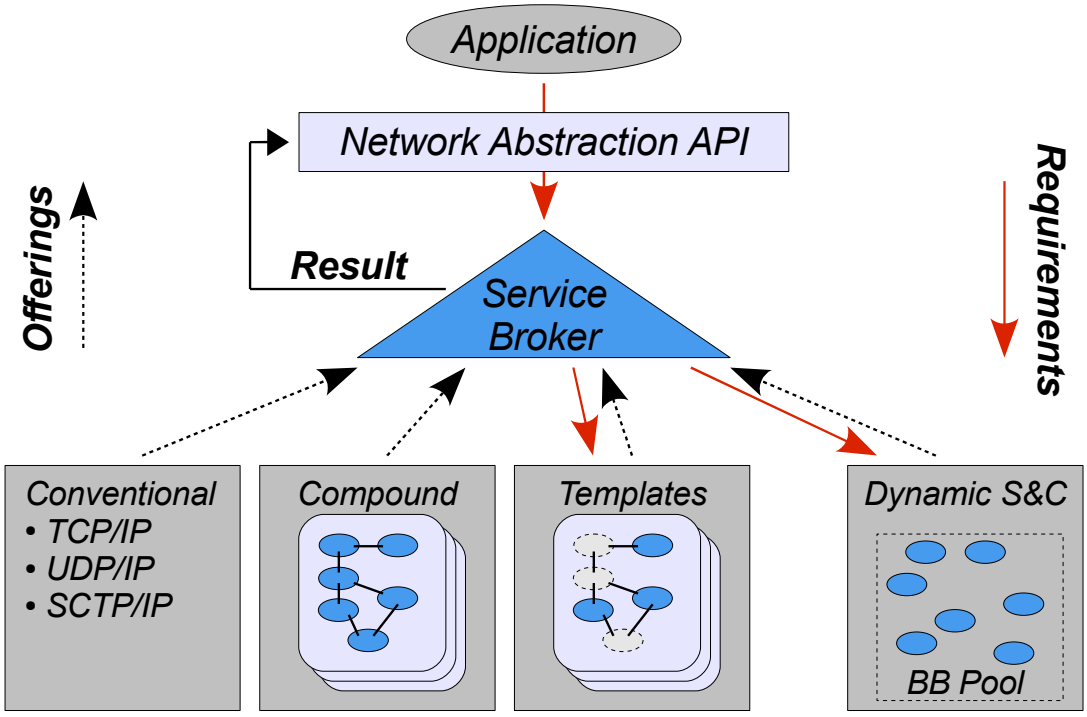


Figure 2.4: Composition Methods and Service Selection [KSRM12]

Composition methods can vary from static- to run-time composition as shown in figure 2.4 where an application designer while using a designing tool does static composition manually. On the contrary, the run-time composition is done automatically after consuming application requirements and, network and other hardware & software constraints.

Partial composition (i.e., a major part of this work namely template-based composition5) method is performed statically at design-time and as well as dynamically at run-time. Different functional composition methods have their disadvantages and advan-

tages; there is a trade-off while selecting a method. Such as design-time composition has less information as it does not have access to run-time information of network and application but as being on design-time, the time is less critical. On the other hand, the run-time composition has more information regarding all the run-time constraints and dependencies, but performs at critical time to compose a communication service. Selecting a pre-generated communication service could be faster to set-up but not flexible, while on the other hand, the dynamic composition takes more setup-time but provides more flexibility.

2.3.6 Rating of Protocol Graphs (PGs)

In case, if more than one protocol graph is satisfying the user/application requirements which one should be chosen? In this case, we need to select the one, which is most nearer (optimal) to application/user demands. However, to compare the values, it is also necessary to normalize the given properties as those can be qualitative (i.e., security) and/or quantitative (i.e., delay, cost and loss rate).

Multiple Criteria Decision Analysis (MCDA) methods like Analytic Hierarchy Process (AHP) [Saa80] can be used to compare the different PGs. For doing this, at first, it is necessary to aggregate the properties of the offered PGs. Then it is required to compare the offered aggregated properties with the properties given as requirements from the application.

2.3.7 Heterogeneity of Services

Networks today are heterogeneous and consist of diverse hardware resources (e.g., Bandwidth, CPU, Memory, etc.), and network policies (e.g., free Internet access for home-users or restricted access from a company LAN). Besides, not all networks support the same functionalities/protocols such as IPv4, IPv6, and SCTP. The flexibility of networks will lead to even more heterogeneity than today. Heterogeneity itself cannot be avoided in general. Heterogeneity itself is not a problem but the attached uncertainty of what is available in the network or at the communication endpoints and what not. This oblivion leads to conservative decisions on ”what” is used. Therefore, any reference model for the FI architectures should support mechanisms for dealing with network heterogeneity at different levels, including running in parallel the various network architectures. Architectures based on functional composition should at least be able to be aware of different network properties and to adapt to differences as far as possible. The network should

also provide information to application so that it can adapt if necessary.

Heterogeneity can be found inter or intra architectures. In inter-architecture heterogeneity, the major issue is that different nodes hold the different kind of protocols that raises the challenges like protocol negotiation and possibly dynamic protocol deployment during run-time. In intra-architecture the challenges are not limited to protocol heterogeneity but also domain constraints by being under different kind of operators. Multiple architectures might be governed by different policies and constraints that raise the challenge of interoperability.

This work does not deal with this specific topic, as heterogeneity itself a major research work. The solution such as negotiation before selection of a communication service and having a common description of service across the networks to come to an agreement about communication-service-selection can be used to solve the heterogeneity problem.

Chapter 3

Service Oriented Network Architecture (SONATE)

This chapter describes the basic principles of Service Oriented Architecture (SOA) and that how they can be used in the Network Architectures. The further, the Service Oriented Network Architecture (SONATE) approach is described, which is developed at the group of Integrated Communication Systems – ICSY[1] within the framework of G-LAb[2]. The presented research work is a part of the SONATE. Thus, it is prerequisite to have an overall understanding of the SONATE.

3.1 Using SOA Paradigm in the Network Architectures

The current Internet is facing the similar problems as Software Engineering (SE). SE has evolved to manage complexities (e.g., maintenance, integration of new functionality, time and task management) of the development process, which has direct effect regarding such as cost, quality and development time. Similar kind of problems is part of the Internet that have not been addressed in the current design principle(s) of the Internet. As the Internet has not been evolved with the change of trends, which made it a victim of increased complexities. To deal with inflexibility and complexity issues of the Internet, we can learn and implement the principle(s) and the technique(s) from the software engineering.

[1] http://www.icsy.de
[2] http://www.german-lab.de/

SE architecture has advanced from structured programming to service orientation paradigm (SOP) [VAMD09] to manage complexities. The design of future network architecture can benefit from SE paradigms to make network architecture more flexible. The SOA approach has already been applied to overlay and grid networks as presented here [Hea05] [Jea08] [BM05]. In this section, it will be argued how SOP can be one of the suitable methodologies for a future network architecture. Before arguing about why to use SOP for a FIA, it will be helpful to describe the fundamental principles [Erl06] of SOA, which are following.

1. Loose Coupling: Coupling refers to the degree of dependencies and bounding between two components. Loose coupling defines independence of a service so that to execute own functionality a service does not require having knowledge of other services.

2. Service Contract: A communication agreement is covered by service description(s) or related documents.

3. Autonomy: The control of a service over its logic characterizes the autonomy.

4. Abstraction: Services are independent of logic they use, and it is hidden from the outside world.

5. Re-usability: A service should be independent and fine-grained enough so that re-usability can be promoted.

6. Composability: An ability of a service to be coordinated with services in a manner that they can form a composite service. Composability fosters re-usability of a service.

7. Statelessness: Property in which services do not keep the state after the request has been processed.

8. Discoverability: A Service should be descriptive enough to be discovered easily.

Most of the issues in the Internet arise because of inflexibility and rigidness of the network architecture. SOP can provide new prospect to build future network architecture. SOP includes the characteristics such as loose coupling, re-usability and autonomy of a service, which are fundamental requirements of a flexible architecture. A Protocol stack can be decomposed into various functionality which are described with formal contract

(i.e., a service description), it makes a functionality autonomous and self-descriptive. A self-descriptive functionality has an ability to be discovered as it carries an attached description, which can be processed by a discovering entity. Abstraction is another point to be taken into account while decomposing a stack into various functionality, it should be at the abstract level where it does not rely on a particular implementation thus, the provided logic should be hidden from a consumer point of view. Characteristics of a BB such as autonomy, description and re-usability, makes it composable. The concept of Composability fosters ease of integration of functionality. Nevertheless, the principle statelessness of SOP might not be appropriate for all functionality of a network architecture as some functionalities of network do require to keep the state such as reliable data transmission.

3.2 SONATE

The Internet architecture is organized as a layered system, where each layer enhances and abstracts the functionality of the lower layers. Interfaces between layers should define the relationship of functionality accordingly. There are no universally accepted evolution principles of the Internet. However, the OSI reference model, which is also a layered system, defines that it should be possible to modify or even exchange the implementation of a layer without the need to adapt to adjacent layers [Rec94].

Thus, there is a need of rethinking network architecture in general [BCSW00]. This chapter discusses the proposed SOA based network architecture. The basic concepts of network architectures based on the service orientation paradigm from layered to layer-less as shown in figure 3.1. The main goal is to develop a flexible network architecture which can adapt to changing requirements and network capabilities as well as it can integrate new functionality easily.

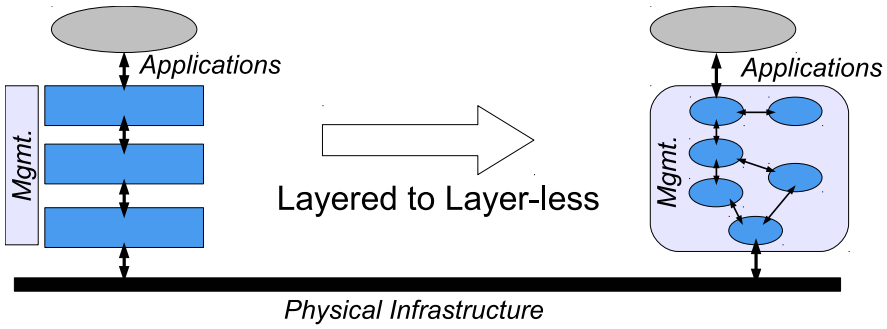


Figure 3.1: Layered to LayerLess Architecture [KSMB14]

The following text describes the overall SONATE architecture before going into the detail of the main components of the architecture.

Web-services implement the specifications of SOA paradigm and are designed for the inter-operation of distributed autonomous functionality on the application level. Network functionalities like routing, data encoding or flow control themselves are inherently distributed. Thus, network architectures cannot use the same implementation like web-services, and because of efficiency issues it would not be appropriate either to process an exhausting meta tagging.

In Web-services, a workflow is generated to fulfill user requirements, and WSDL is used for interface specifications. However, web-services implementation does not define how to describe the service semantics. Where interfaces help to check the comparability of different web-services with each other but they do not describe what is being covered in a web-service. So an automatic composition will not have an idea that which web-services is to combine.

In this architecture, not only that ports define interfaces, but also which effects are provided and required by those ports. Hence, by the use of the port concept, the semantic of a service is covered which can be utilized for an automatic composition.

The interfaces of a building block should reflect the provided effects and hide the implementation details.

In the SONATE [MR08] [RH08], a service oriented network architecture is presented. SONATE aims at supporting dynamic composition of communication services by generating dynamic protocol graphs[3]. Without being dependent on a static protocol graph, it is easier to make use of new building blocks and to reuse functionality on different levels. Having dynamic protocol graphs implies that there is no static placement of functionality as defined by the layers of the OSI reference model. In this sense such networks will be *layerless* such as compression or encryption can be used for application payload only or also for some protocol headers. Furthermore, it is not necessary to process the protocols in a sequence. For instance there might be different branches in a protocol graph to handle different but related data flows (e.g., signaling and streaming media) at the same connection. Service description is used at the early stages of composition to enable the flexibility.

Figure 3.2 shows the SONATE approach where an application sends the requirements 3.2.1 via requirements based API 3.2.2, which are received by a service broker. Service broker sends the requirements to a composition process 3.2.4 where the protocol graphs

[3]A protocol graph corresponds to workflows in SOA terminology.

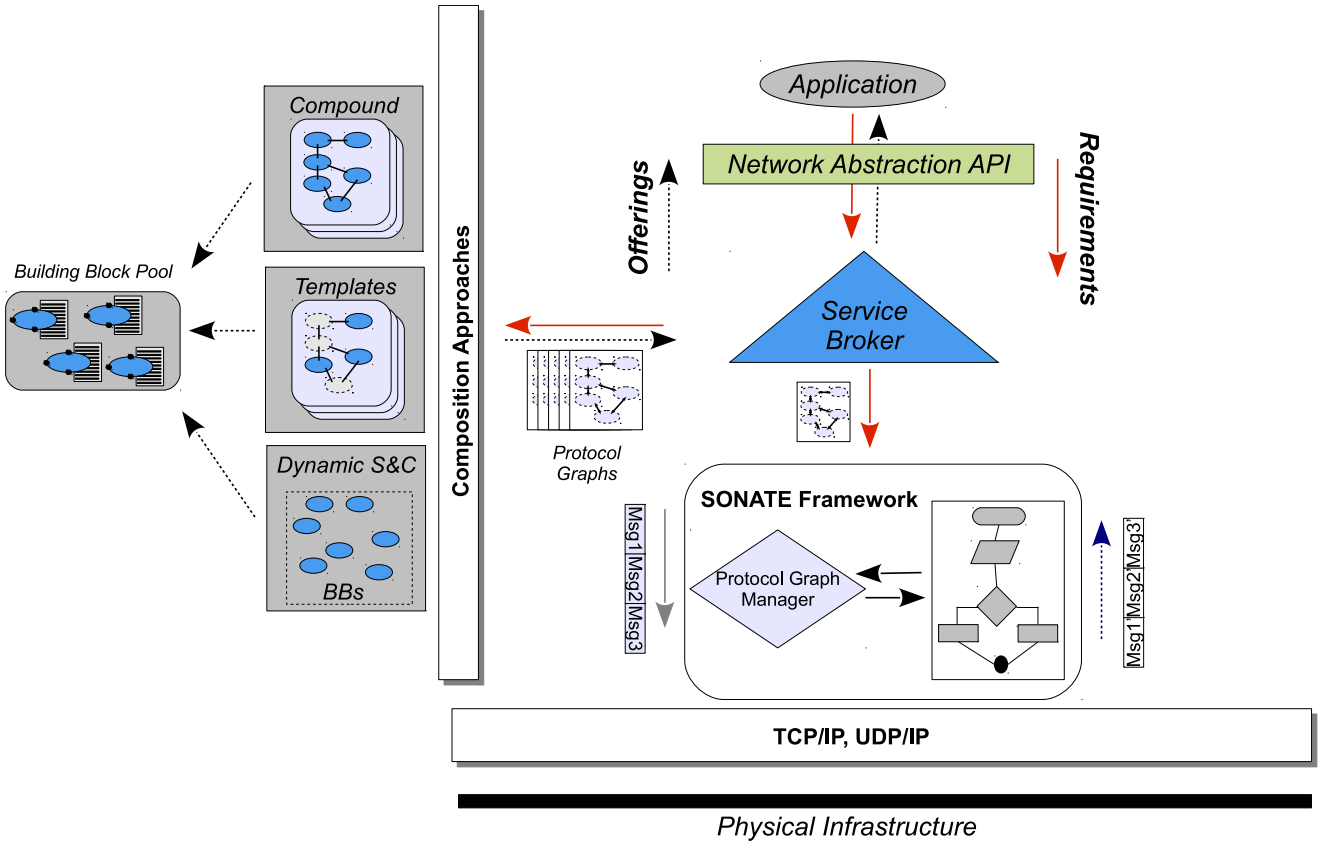


Figure 3.2: Service Oriented Network Architecture (SONATE) [KSRM12]

will be generated. A composition process may generate more than single protocol graph so that the most suitable one will be selected. A selection process 3.2.6 runs inside the service broker. After the selection of a protocol graph, it will be directed to the SONATE framework 3.2.7 for execution and to initiate a communication.

The following text describes the requirements description, API, template-based composition process, AHP based service selection process and execution framework respectively.

3.2.1 Description of Requirements

Requirements are consisted of two parameters (i.e., service-name, user-requirements) which are needed by an API. Where the naming parameter specifies what service is accessed, and the user-requirements parameter specifies how. Requirements from a user or an application and offerings from the network are represented by a quadruplet of effect, operator, attribute, and weight as shown in fig.3.3 [KVM11].

Operators link effects to attributes, typical operators are "<‘‘, ‘‘>‘‘, ‘‘=‘‘, " <=″, etc. Attributes quantify or qualify the effects. They represent values like an exact quantity (e.g., $delay < 20ms$), Boolean values (e.g., $Packetordering = true$) or a quality (e.g., $delay = low$). The qualitative parameters need an extra mapping or a domain-based taxonomy. There is an additional branch of weight that is used in specifying requirements. By using the weight attribute, an application or a user can emphasize on the importance of particular requirements.

Requirements and offerings are structured in a flat scheme and do not have any parent-child relationship like having multiple requirements under single requirement (i.e., to fulfill single requirement, a composition process must fulfill other sub requirements). The benefit of this scheme is that an application developer does not need to worry about the place of a requirement. Any new requirement can be added without extra knowledge about dependencies (i.e., parent-child relationship) among other requirements. However, this might not be an economical choice in many cases such as in a selection and composition method where every single requirement is not dealt independently. In this case, two interdependent requirements must be processed together such as to select an encryption-BB the process must consider the functional requirement of "Ciphering=true" and as well as the key strength (e.g., Ciphering.key=256).

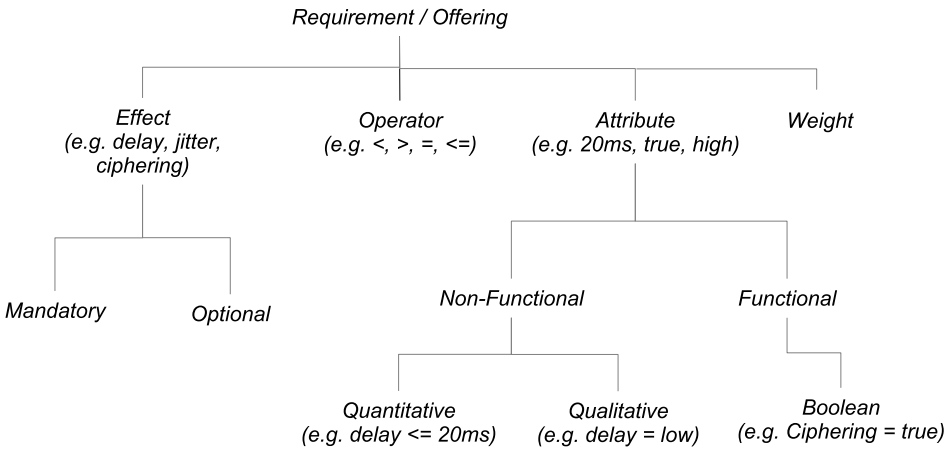


Figure 3.3: Requirements/Offerings [KSMB14]

3.2.2 Requirements-Based API

The design concept of socket API allows defining address family. However, it relies on applications to specify the exact protocol to use. The current socket API does not

hide underlying protocols from the applications, which makes it harder for an application to switch from one transport protocol to other transport protocol such as from SCTP [SXea00] to DCCP [FK06], as an application needs to specify the type of a protocol exclusively. The exact demand of a protocol by applications fosters tightly bound coupling, which forces the underlying network to use that exact protocol rather than an improved version of a protocol or one that is more suitable for the given network conditions. To deploy a new protocol in the Internet architecture, it is not enough just to change the application but it also requires modifying or exchanging the API. The tight coupling also causes the issues [KWL+10] like compatibility (e.g., IPV6 has no backward compatibility to IPV4), Middleboxes (i.e., rejection of packets in case of unknown format of a packet), and Operating System (i.e., adaptation of deployed protocol in vendor OS).

An application needs to be modified if it wants to use a different transport protocol than currently being used such as an application intends to shift from TCP [oSC81] to UDP [Pos80]. Peer addressing and address-resolution are part of an application, which makes an application address type dependent. Thus an application has to select a particular addressing type such as IPV4 or IPV6 or a mapping/translation mechanism (e.g., Application Layer Gateway [vB11]) must be deployed at lower layers. Different addressing types require different socket structures (struct) so that there is a difference between IPv4 TCP socket structure and IPv6 TCP socket structure and same is true for UDP.

The call of "setsockopt" is an example of another dependency where protocol specific options such as TCP_NODELAY can be turned on or off, as options are specific to a protocol so that it is must for an application to know details about a protocol.

If an application needs to switch transport protocol, it is not enough just to adjust socket options or to change addressing family but it is also required to alter the existing API(e.g., Sockets API Extensions for SCTP [STP+11]) or to use a different API.

Abstraction is used for hiding complexity and to encourage flexibility. In this approach, it is proposed an API by which an application can send its requirements in an abstract form to the underlying network in a way that applications do not need to rely on specific protocols; the process of selection would rather be handled by the network architecture. Network architecture should able to process those abstract requirements of applications. Abstract requirements from the applications also help to create a unified API so that a single API can be used for multiple means of transport protocols transparently to foster a loose coupling between the application and underlying protocols.

Current applications are tightly coupled with the given protocols, though they only care about its connectivity demands are fulfilled. A Requirements-based API [SM12]

[LMW+11] will alleviate the developer from choosing a protocol. Instead, *requirements* will be communicated to the underlying network architecture. An application uses these requirements to request specific characteristics of a connection. Requirements are specified regarding effects/capabilities, an effect is a visible outcome of a functionality such as flow control functionality provides effect of transmission rate adaptation between two parties.

A requirements-based API has been proposed, the further details can be found in this publication [SM12].

3.2.3 Building Block Description Language

The description of a BB2.3.2 contains the information such as covered effects and the interfaces that are needed to connect a BB to others.

Figure 3.4 shows the schema of a building block description language. Every building block has a unique universal (i.e., UUID) and building block id (i.e., BBID), BBID is used internally by a composition method to distinguish BBs in a protocol graph so that execution framework can select a right BB instance out of BB pool. The schema consists of two major branches so called "Port" and "Optional" as shown in fig.3.4. The "Port" section deals with the ports of a BB, ports work as communication end-points in BBs interaction [SGKee11], the section ports is the most crucial part of the description as each port describes the offerings and the requirements on it. Moreover, accumulation of all the offerings on the ports defines the effects covered in a building block. Besides, revealing the covered effects, ports are used for connecting a BB to other BB(s). A composition method matches the port offerings (i.e., output) of a BB to the requirements (i.e., input) of other the other BBs to connect the building blocks together.

Offerings and requirements of a port have a same internal structure consists of a triplet of effect, operator and attribute so that they can be compared to connect to other port. The triplet is described in detail in the sub-section 3.2.1 except weight. The "Optional" branch of the description language covers the properties of a building block, which in the later process summed up to the same properties of other BBs to select the most suitable communication service.

3.2.4 Composition of Protocol Graphs

Once the application requirements are known, a composition process starts by checking the requirements and the available effects (i.e., covered by the BBs). To compose a

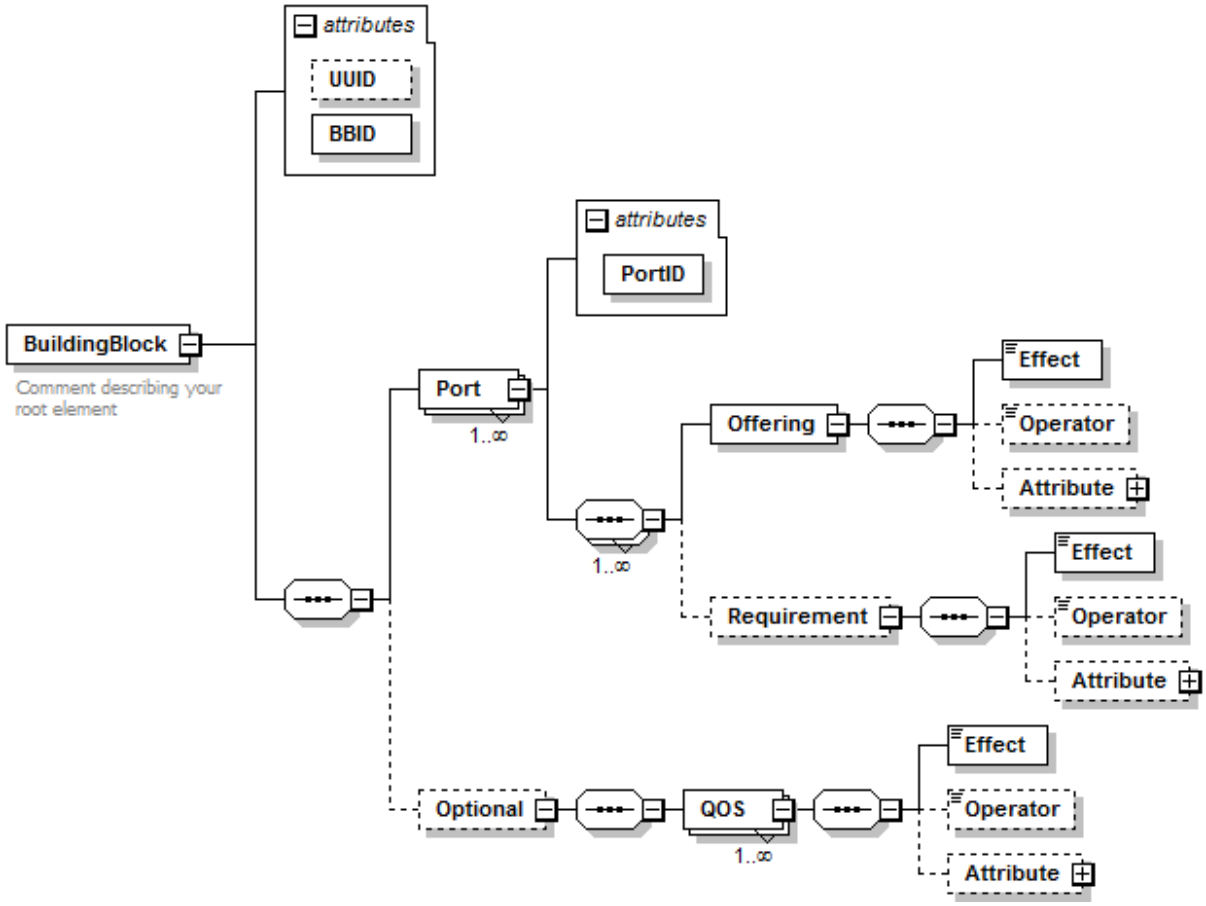


Figure 3.4: Building Block Description Language Schema

protocol graph, first the suitable building blocks are selected from the BB pool, there could be more than one building block that would satisfy the application requirements. After the building blocks have been selected, they must be placed (i.e., ordered) properly so that there are no conflicts. Besides, it should create an executable protocol graph. Finally, the building blocks are connected to each other via ports as described in the previous section.

The composition methods differ in terms of the time phase it uses to generate a protocol graph. A composition can be performed at different phases (as described in 1.1) which vary from development-time to on-the-fly (i.e., runtime).

Design-time composition creates a static protocol graph, which cannot be changed at the later stages, whereas Runtime composition provides the most flexibility. Both design and run-time compositions have their own advantages and disadvantages some of those trade-offs are discussed in [SM11].

Based on arguments given here and 1.1, an innovative composition approach so called

template-based composition is presented in this work. Template based composition is a partial run-time approach; Template based approach is the main topic of this work so that it is described in the separate chapter 5 in detail.

3.2.5 Protocol Graph Description Language

After a composition process has been executed, it will generate a single or multiple protocol graphs. However, a composition process must create a protocol graph that can be interpreted by the intended execution framework (i.e., in this case, SONATE Framework) to render the requested communication service from an application. Hence, it is necessary to have a description language, which is known to the composition process and to the execution framework (i.e., to execute a PG). A protocol graph is described in a specific language. The description language is based on XML. The schema is shown in the figure. 3.5.

The description language is divided into three major elements named as building-blocks, connections, and offering.

The ”Optional” branch of the description language is used in a selection process of a protocol graph, in case there is more than single protocol graphs have been generated by a composition method. The selection of a protocol graph and use of the ”Optional” section is described in the following sub-section 3.2.6.

The ”buildingblocks” part of the language tells an execution framework about the included building blocks in a protocol graph so that they can be retrieved from the BB pool and instantiated. Its ID or UUID can identify a BB depending on the scope of the retrieval. Whereas, the ”special” attribute symbolizes the two exclusive building blocks namely ”app” (i.e., application) and ”net” (i.e., network), which are responsible for collecting/dispatching the data from/to application and network respectively.

The ”connections” branch in the description language defines the ordering of building blocks and ultimately execution flow. Whereas, ”blockname” & ”blockid” attributes are related to building block and ”name” & ”id” attributes describe a port.

3.2.6 Protocol Graph Selection

Once a composition process is carried out, it may generate more than single suitable protocol graphs. It is now necessary to select one out of them so that it will be used for the communication establishment. *If any of the generated protocol graphs can be used for the communication, the simplistic selection mechanism will be to select the first match.*

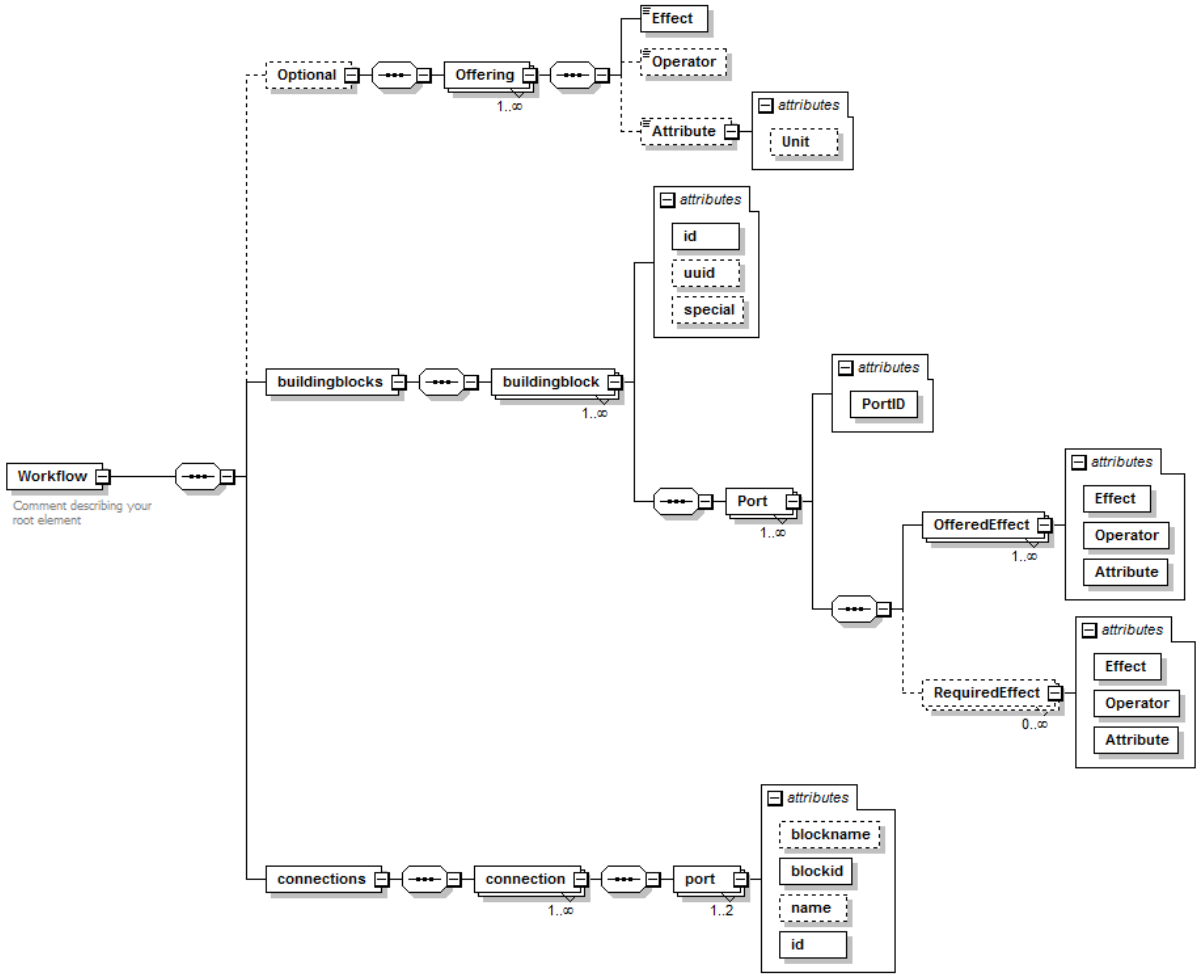


Figure 3.5: Protocol Graph Description Schema

However, it is in the best interest of the requested application to find the most optimized match among the generated protocol graphs. To extract the most optimal match, the qualitative and quantitative properties (e.g., delay, jitter, bit-rate, throughput) of an individual protocol graph are used as a selection criterion. Before these properties can be compared to find the most appropriate protocol graph it is needed that the application/user defines that which criteria are more important by assigning the weight to the individual properties. If no weight is specified then it is up to the selection process to deal with it.

The selection of a PG uses multi criteria so that it can not be solved by a simple comparison algorithm. Hence, Multiple Criteria Decision Analysis (MCDA) methods are used for the selection, two of those methods are Analytic Hierarchy Process (AHP) [Saa80] and Multi-Attribute Utility Theory (MAUT).

3.2.7 SONATE Protocol Graph(s) Execution Framework

The SONATE execution framework provides a mean to execute protocol graphs. In execution of a PG, a BB is invoked and executed; the corresponding values (i.e., resultant after BB execution) are passed as input parameters to the connecting BBs. The basic concept of SONATE Framework is described in this paper [RSS+09]. The SONATE framework deals directly with the instances of building blocks to provide the required communication service. A protocol graph is described in the Extensible Markup Language (XML), and the SONATE framework can process an XML based protocol graph. The major tasks in the processing of a protocol graph involve retrieval of BBs from the repository, the connection of BBs as specified in the PG, execution of BBs and exchange of data/information among the building blocks.

The SONATE framework has a JAVA-based implementation. To enforce loose coupling, the BB interaction model hides all BB's implementation from each other. Since building block instances still need to communicate with each others to solve this problem the concept of named communication endpoints called ports is introduced. Building blocks can use the ports to distinguish different kinds of communication partners or different operation modes, the concept of the ports is discussed in detail in this paper [SGKee11].

3.3 Contribution

The SONATE is a combined effort of G-Lab members at ICSY. However, every individual member had a specialized focused within the project. "Dennis Schwerdel"[4] worked on a run-time composition approach [Sea10a] and the port-concept [SGKee11]. "Rahmatullah Khodoker" [5] proposed a service description [KVM11] and PG selection [Kea12] method. Whereas, "Daniel Guenther" [6] researched on a run-time selection and composition approach [Gea11].

Other proposed approaches are performed at run-time, whereas the presented work suggests a novel intermediate composition approach which exploits the strength of different time-phases to generate the protocol graphs with reduced time-complexity and yet flexible enough to adapt to application and network requirements.

[4] https://www.researchgate.net/researcher/61636997_D_Schwerdel
[5] https://www.researchgate.net/profile/Rahamatullah_Khondoker
[6] https://www.researchgate.net/profile/Daniel_Guenther/

Chapter 4

State of the Art of Functional Composition

This chapter describes the state of the art on functional composition to have a better overview of the topic. The presented approaches are divided into categories on the time-phase. The section "Design-Time Composition"4 covers the projects with static composition such as RNA [TWP06], SILO [DRB+07] and, NetServ[1]. Whereas section "Run-Time Composition"4.6 describes dynamic composition like RBA [BFH03], DaCaPo [VPPW93a] [VPPW93b] and, Network Service Architecture [GW07] approaches in the network architectures. The section "Intermediate Composition"4.14 covers the most related work to the proposed approach of Template-Based composition5, as the proposed approach also deals with intermediate composition method by splitting it in different time-phases. The related work in this section is not only taken from network architecture but also from the web-service domain as this strategy is not yet implemented in many network-based projects. Further, intermediate composition approaches are compared to each other and then finally to the Template-Based composition.

Design-Time Composition

4.1 Net-Silo

Service Integration Control and Optimization (Silo) [DRB+07] [2] is a US NSF project that also uses elements of fine-grained functionalities as reusable building blocks. Their main

[1] NetServ Project - http://www.cs.columbia.edu/irt/project/netserv/
[2] Net Silos http://net-silos.net/

objective is to separate control from data functions to enable cross-layer interactions. A service is fully defined by describing the function it performs, interfaces it presents to other services and its control parameters that are also referred as knobs. Knobs (service parameters) can be tuned dependent on the application requirements. A method is an implementation of a service that uses a specific mechanism to carry out the functionality associated with the service. For instance "re-sequencing" is one method for implementing the "in-order packet delivery" service. A silo is a dynamic composition of building blocks, which are instantiated on a per-flow basis. A control agent, available at each node, is responsible for the composition of a silo. The composition refers to determining the subset of services, their order in the stack, and the service implementation. Besides, composition process also adjusts all service & method knobs based on the requirements and the available resources. An ontology, constraints and dependencies among services, has been established for the fine-grained composition. In [VWR+07] the authors present a simplistic service composition approach for the SILO network architecture. It is a recursive approach based on pre-defined (at design-time) precedence constrains of services.

4.2 NetServ

The NSF NetServ project [3] started in Spring 2009. It follows a bit different approach to ANA and RBA functional composition. Whereas in ANA and RBA, the nodes functionality is dependent on the packet header and content, in NetServ the service invocation is signaling driven. The implementation is based on the Click Modular Router [4] which is an OpenSource C++ project. Services are deployed in the OGSI Java Framework. OSGI supports loading and unloading of building blocks at runtime that can be simple bundled jar files. This setup enables dynamic in network service deployment.

The NetServ project concentrates on the implementation of Content Distribution Network (CDN) enabled by additional services implemented on top of the Click Router.

4.3 Network virtualization architecture (VNet)

In the current EU Project 4WARD [5], the virtualization of network resources is investigated. Network virtualization allows the concurrent operation of multiple network types

[3] NetServ Project - http://www.cs.columbia.edu/irt/project/netserv/
[4] http://read.cs.ucla.edu/click/
[5] 4WARD http://www.4ward-project.eu/

on the same infrastructure thus providing a multiple instead of an IP "one-size-fits-all" solution for networking. In this project, virtualization takes the central role and is considered as a technology for enabling Internet innovation [SWP+09]. The idea is to have different virtual networks, which can be built according to different design criteria and are operated on the custom-tailored network. There are four main players in the proposed approach namely Physical Infrastructure Providers (PIPs), Virtual Network Providers (VNPs), Virtual Network Operators (VNOs) and Service Providers (SPs). Where PIP owns & manages physical infrastructure, VPN is responsible for composing the virtual resources, VNO provides installation & operation of a VNet and SP uses VNet to offer its service. The virtual path is connected beforehand, later, any user or application can request to join a virtual network.

4.4 Node Architecture

The node architecture is another project that has been carried under the EU project of 4WARD. This project [VME+09] also uses the virtualization for running the different virtual networks in parallel. A so-called Netlet [VMW+09] contains a collection of functional building blocks and can be seen as a container i.e., to encapsulate today or a future network protocol stack. Netlets hide the protocol details and the only way to communicate is via provided interfaces. They can be exchanged without the need to change the application or network interfaces. A Netlet Selector depending on the application requirements can choose such a Netlet. The functional building blocks can be reused in different Netlets, and are stored in a Repository. These Netlets are created at design-time but they also contain knobs for configuring the certain properties of the Netlets at run-time. Architecture specific multiplexers are used to run the Netlets in parallel.

4.5 RNA

The Recursive Network Architecture [TWP06] uses a different approach by introducing a meta-layer which is a generic protocol layer offering essential services. Each protocol layer stack will then be instantiated from this meta-layer, which provides cross-layer interactions. The meta-layers are designed at the design-time. The goals are set at the run-time to configure the meta layers based on the requirements. Project provides the limited information. Hence, it is not possible to know the exact implementation details.

4.6 AutoI

The Autonomic Internet, AutoI [6] is a running European Union (EU) project which targets a self-managing communication resource overlay for secure, reliable, guaranteed and fast delivery of services. For achieving this objective, AutoI is focusing on designing and developing an open source framework for the composition and execution of reliable & guaranteed services. For doing so, it merges virtualization of network resources and policy-based management techniques to define, describe and control the internal logic of services.

Run-Time Composition

4.7 Role-Based Architecture (RBA)

RBA was proposed in [BFH03] and can be seen as an abstract approach to a non-layered architecture. The RBA is organized in standardized building blocks, which are called roles. Each role has its roleID that reflects its functionality. In the initial concept there was only a small and limited amount of roles considered. There can be multiple roles on a single node and a role can also be abstractly distributed over multiple nodes.

RBA decomposes the network stack and therefore the packet header structure will no longer be a stack but a heap of headers. This use of Role Specific Headers (RSH) is the main concept for role interaction and composition in RBA.

The RSH semantically define the inputs and the outputs of a certain role. They address roles at specific nodes in the way roleID@nodeID; in case of the distributed role this may also be roleID@* which means that each node along the path with that roleID will process the packet. Along with its path the packets RSH can be read, modified, added or deleted. A role may also contain an internal role state that can be changed by signaling through another RSH or can modify its activity depending on the set of RSH in a packet.

How this internal state can be used is shown by the example of forwarding. RBA does not provide a model for routing packets, but for header processing. In a packet from A to B, a role specific header will address the "Forward" role with additional information Forward@*, sourceID, destinationID on every node. How the forwarding is done will be determined by another role that first calculates the routing path and then it sends

[6] http://ist-autoi.eu/autoi/

another RSH to change the state of the Forwarding role.

When a packet arrives, the node must decide in what order the roles must be executed. Because some functionalities require specific ordering (e.g., Compress, Encrypt, Expand, Decrypt is not useful), there also needs to be some precedence information carried in packets. Roles that communicate directly with each other on one node should be composable into an aggregate role. These roles are directly bounded instead of using shared data in RSH.

The authors especially emphasize that RBA is a generic model, in the sense that it can be applied on top of any layer, whether all network functionality is split into roles or only the functionalities above the Link, IP or even application layer is decomposed, depends on the realization.

4.8 Self-Net

Self-Net (Self-Management if Cognitive Future Internet Elements) [7] is an ongoing European Project that investigates approaches for autonomic self-management of the network stack. Functionality is also split into functional blocks (network elements), which form network compartments. The orchestration of network elements and the selection of network compartments are achieved through cognitive cycles (measurement, execute, decide) which empowers self-organization and optimization. As the project is still young only preliminary results are available.

4.9 Automatic Network Architecture (ANA)

The project ANA [8] has been funded by the EU commission for a three years period since 2006 realized the first implementation of a non-layered network architecture. In ANA, the functionality is decomposed in functional blocks that perform the processing. Functional blocks can have arbitrary granularity such as the size of a small entity or a whole network stack. Functional Blocks can be accessed via one or multiple Information Dispatch Points (IDP). Data packets are always sent to a certain IDP not to a functional block itself. Having multiple IDPs for one FB enables the differentiation between different functions or states that are performed by the functional block. If IDP A is accessed then

[7] Self-Net https://www.ict-selfnet.eu/

[8] Autonomic Network Architecture (ANA) http://www.ana-project.org/

the function performs forwarding to an address A whereas when IDP B is used the information is forwarded to a different address B.

An Information Channel (IC) is a functional block that is also accessed by an IDP and provides an information channel to some remote entity. ICs can be a cable, radio medium or memory, but can also be a composed service that offers multicast or broadcast. ANA uses the term compartment for different network "instances" which includes nodes with the same functional blocks. One of these compartments may also be legacy IP, but a compartment can also be bounded by different network technologies (wireless, wired). Each compartment can have its own communication scheme (incl. addressing, forwarding, etc.). Nodes can be part of different compartments as long as they include the functional blocks for these compartments. These "multihome" nodes can act as gateways for translating the communication between the compartments.

An ANA Node includes the MINMEX (Minimal INfrastructure for Maximal Extensibility) and the playground. Whereas the playground includes all functional blocks, the MINMEX provides packet forwarding between different functional blocks on a node and access to other protocols and compartments.

Each functional block knows a successor IDP and, after processing, sends the data to this IDP. Dispatching from IDP to IDP or FB to FB continues until an IDP is called which provides a low-level function, which forwards the packet the next node via a network interface. Different functional compositions are achieved by manipulation of the IDPs either bounding the IDPs to different functional blocks, deleting or adding them. For loading functional blocks and exchanging service specification ANA uses the Active Service Deployment Protocol [SSCH03], which can query the presence of services such as (un-)load, (re-)configure, (de-)activate services and service composites. The composition of atomic functional-blocks into higher-level service components is done by the use of service control graphs and filters [Sif08]. The control graph is a cyclic directed graph and represents all possible processing paths through functional blocks whereas the actual processing path is runtime determined. The actual processing path depends on the content of the message, which is matched by message filtering rules and external events that are used for autonomic adaptation.

A first running implementation of ANA can be used as standalone user-space application or as Linux kernel module. ANA has not been designed for performance more as a proof of concept. The implementation provides basic functional blocks, but also provides interoperability with legacy IP.

4.10 Coyote

Coyote [BHS+98] is motivated by the challenges of mobile computing systems like service disconnection, location and QoS support. The required functionality for mobile hosts and base stations are structured in protocols which can be implemented in a variety of micro-protocols. In Coyote, composite-protocols and protocols are composed in hierarchical order whereas micro-Protocols are composed in non-hierarchical to provide more freedom to composition. The micro-protocols are structured via events and event handlers in order to reduce interdependencies among them. The actual micro-protocol selection is run-time based, which depends on the events that are originated by the system or the micro-protocols.

4.11 Network Service Architecture

With [Wol06] [GW07] [SW08] [HGW08] [SHPW09], the authors present the series of publications that present an automated network service composition approach and implementation. In [GW07], they propose a first design for a network service architecture. A service controller (i.e., one for each autonomous System, organized in a hierarchy) manages some network service nodes and upon connection request sets up the service processing sequence and the data transfer between the service nodes for each flow. The request is then passed to the neighboring service controller along the path to the destination so it can further setup the connection. This implies that fixed routes for each flow are assumed i.e., connection-oriented packet switching. Each service node has to maintain state for each flow and each packet of the flow is processed equally.

The Service Controller receives the user request and parses the information for a Mapping Algorithm. The Mapping Algorithm uses service and resource information (i.e., available services, memory, power, bandwidth from the service nodes) and tries to map the user request to service nodes. In [HGW08], they present a distributed service routing algorithm to find an (nearly) optimal network path that incorporates all required services. A Service Node Interface allows the Service Controller to send control messages for connection setup to the network service nodes.

Initially a service node registers its available services at the service controller. It then receives a connection request from the service controller and keeps track which flows require which services. It receives, processes and transmits packets for each flow accordingly - based on a UDP hop-by-hop connection. The Service node monitors resources which it

sends to the Service Controller.

In [SW08], they describe and implement a "service socket API" which provides end2end abstraction like the commonly used BSD TCP/IP sockets. The API consists of the following three methods:

- Method-1 is a request function that takes as input the required service specification then sets up the connection and finally returns a socket.
- Method-2 uses returned socket to create packets and to send the data to the next node.
- Method-3 receives and stores the arriving packets in the memory.

In [SHPW09], they further describe an automatic service composition approach based on the semantic description of data and communication characteristics with the Web Ontology language (OWL). They structure typical network characteristics in a tree hierarchy that can be expressed in OWL. Based on preconditions and transformations they formulate the service composition approach as an AI planning problem. The AI problem is formulated in Golog, a high-level programming language which is based on situation calculus which is a logical language for reasoning. They have shown the operation of the network service architecture on the EMULAB testbed and a prototype implementation on the Cisco ISR 2800 router, which provides an Application eXtension Platform (AXP).

4.12 Dynamic Configuration of Protocols (DaCaPo)

This approach [VPPW93a] [VPPW93b] proposes an architecture for dynamically configuring protocols based on QoS requirements and Network Coding. The DaCaPo infrastructure is divided into 3 layers - Transport, Application and Middle C layer where the DaCaPo protocols are processed. The composition is based on an acyclic graph that shows the order of protocol invocation. The selection of the direct sequence depends on the application requirements. The sequence is then confirmed with the receiver and the modules initialized.

The model is realized by three cooperating entities where configuration process selects the most suitable protocols, the connection manager negotiates the configuration, and the resource manager provides the run-time environment. The protocols are configured at run-time on application requirements and QoS parameters.

4.13 The Function Based Communication Subsystem (FCSS)

FCSS [Sti94] shows the decomposition of tasks in functional modules, which should be aligned in a stack but based on the application requirement. In [GR97], the authors point to a drawback of the above approaches and ask for a generic description so that the new deployment can be facilitated and the customization of the implementation is kept at a minimum. They propose the use of the Service Description Language (SDL) which is the predecessor of the current Web Service Description Language (WSDL).

In [SSSZ94], the authors present a protocol-machine-configuration and a protocol-resource-descriptor languages that support a function-based approach to creating custom application based protocol machines. The different data streams are supported for those protocol machines. A scenario of collaborative distance learning has been used to show the functionality of presented languages and the approach.

4.14 TARIFA

Tarifa [SLJJ+09] is a clean slate approach that targets an architecture covering aspects of ubiquitous service provision besides its Composition. Services are classified into atomic and composed services. Atomic services are those individual functions commonly used in networking protocols (i.e. flow control, sequence numbers, etc). Composed services are network applications with a wider scope (e.g. printer service, directory service, instant messaging, etc). Composed services are likely to build upon atomic services and/or other composed services. Services are considered to be distributed which could be discovered and executed while passing through the route. In this approach, different negotiation schemes (i.e. 3-way and 4-way handshake) have been proposed to negotiate application request for the desired network functionality and QoS requirements. Tarifa also uses the run-time to compose the composite services. However, the considered domain has the small number of atomic services so that it can be easily handled at run-time.

Intermediate Composition

4.15 Semantic-Based Semi-Automatic Web Service Composition (SBWComp):

The main purpose of the project [AP10] [AP11] is to assist non-expert users, who are oblivious about how to achieve their goals to acquire the optimized solutions by composing a web service and to satisfy their requirements. The approach uses domain information and semantic to cover a specific domain. The generic process templates are proposed to users to achieve goal above. A process template consists of the workflow (i.e, which is composed of several functions) linked with control flow, structured activities (e.g., loops, conditional statements) and user preferences. There are no examples of template in the publications, however, in [AP11] it is stated that: "'The generic process template acts as a configurable module. It defines generic participating activities and the control flow. We are still investigating the best way (language) to describe such templates. Indeed, this question will be one of our major research focuses"'. A smart home (i.e., equipped with various sensors) scenario is selected for the proof of concept. The scenario explains different service orchestrations for saving the energy in a smart home, in addition to, fitting to one's living habits and circumstances.

4.16 A Template-Based Mechanism for Dynamic Service Composition Based on Context Prediction in Ubicomp Applications (DTSComp):

This approach [MHK07] covers the pattern-based composition in an intelligent living scenario. The approach uses a list of predefined tasks corresponds to specific actions that lead to a peculiar service pattern. The used services in the patterns are also preselected which restricts it from using the different implementations of the same service.

4.17 Rule-based semi automatic Web services composition (RSWComp):

In this approach [ZOP09], a user performs the service composition with the help an easy drag and drop tool (i.e., similar to the Mashup tools like Yahoo Pipes). The composed service is later translated into an intermediate language, which can be initiated and executed by the system to provide the requested service to the user. In the proposed approach the defined workflow uses a lower abstraction as input and output variables are part of the workflow description, which consequently makes it dependent on the particular implementation.

4.18 Dynamic Reconfiguration Using Template Based Web Service Composition (DTWComp):

In [GSM08], a template based approach is used to deal with re-usability issue of WS-BPEL. It proposes a framework that allows the design of WS-BPEL processes in a modular way. The concrete modules of WS-BPEL activities are modeled as templates and stored for the re-usability. The approach [GSM08] combines different templates to create a workflow. A set of templates works as a pattern that can be reused for different scenarios. The composition of the workflow has not been specifically discussed or described in the presented approach.

4.19 Pattern Based Composition of Web Services for Symbolic Computations (PBWComp):

In [CMPK08], a dynamic but yet design-time workflow composition approach is presented which utilizes standard workflow patterns to solve the symbolic computing problem in the context of web services. In this approach, the general workflow patterns are used for helping Computer Algebra Systems (CAS) user(s) to describe the relationships and the sequence of service calls.

4.20 Semi-Automatic Composition of Web Services using Semantic Descriptions (SWSComp):

In [SHP03], a semi-automatic approach is presented to guide the user in a dynamic service composition process. The semi-automatic process presents the matching services to the user at each step of service composition. The approach uses the semantic description of the services to evade the barrier between human and machine understanding. The developed prototype consists of two basic components so called a composer and an inference engine. The inference engine stores the information about services in its Knowledge Base in addition to finding the matching services for the user. Moreover, the composer handles the communication between the user and the engine. To find an exact match, the inference engine looks in the ServiceProfile description for the same OWL class. The priority of the suggesting matches is ranked lower as the distance between the two types in the ontology tree increases. In case, the presented choices are in an excessive amount, further, filtering is performed based on non-functional attributes such as location, type, deployment data, and sensitivity to reduce the number choices, so that, the selection process for a user can be eased. After selection, execution of composite service is performed by invoking the each individual service and passing the data between them.

4.21 Automatic Composition of SemanticWeb Services (ASWComp):

In paper [KBGH07], a semantic-based automated web-services discovery & composition approach is presented. A service is defined by 6-tuple (i.e., S= (CI;I;A;AO;O;COi)) namely preconditions, inputs, side-effect, affected object, outputs, and post-conditions. A query is used to formulate the requirements, and based on the query a service is looked up in the service repository. Requirement query is described by input parameters, output parameters and conditions. A query is satisfied if one of these requirements is satisfied: (i) it produces query output parameters and satisfies the query post-conditions; (ii) it uses the provided input parameters and satisfies the query pre-conditions; (iii) it produces the requested side-effects. The composition is based on automatically finding a directed acyclic graph of services to obtain the desired service. The semantic of the services were not presented. Hence, it is unclear how composition process can match the unrelated input and output parameters of individual services.

4.22 A Service Composition Method Based On The Template Mechanism In The Service Scalable Network Framework (TNSComp):

In [FS13], a redesign of the network framework below the application layer is proposed. A service container is used instead of the multi-layer structure of the network architecture. The service container encapsulates the network function such as scheduling, queue operation, QoS control. A service container provides the essential services. The network services are divided into three categories namely connection management services, QoS management services, and network security services. Moreover, the templates are used for configuring the individual protocol units (i.e., instances of the services) to provide the required service. At first, a formal description of a service template is written, and then a parser parses the description to load the appropriate protocol units into the service container. The approach is quite recent as presented in 2013. The data flow and the connections among the services are not covered which are the crucial parts of the composition.

4.23 Semi-Automatic Distribution Pattern Modeling of Web Service Compositions using Semantics (SPWComp):

In [BP06], a distributed web service composition approach is presented, which relies on semantic description and user involvement much like the approach presented in [SHP03] with difference of being distributed rather than being centralized. Web services are described in OWL-S to generate the distribution pattern model. A tool is implemented so-called TOPMAN (TOPology MANager) to do the modeling and to compose a web service.

4.24 Comparison of Intermediate Composition Approaches

A comparison of the approaches above is shown in the table 4.1 to have an overview of the work. The approaches are compared by time-phase, human involvement during composi-

Approach	Automatic \Manual	Runtime \Designtime \Intermediate	Static \Dynamic \Configurable	WS \Network	Distributed \Local
DTSComp 4.16	Automatic	Runtime	Static	WS	Local
RSWComp 4.17	Manual	Designtime	Dynamic	WS	Local
SBWComp 4.15	Semi	Runtime	Dynamic	WS	Local
DTWComp 4.18	Semi	Designtime	Dynamic	WS	NA
PBWComp 4.19	NA	Designtime	Dynamic	WS	Local
SWSComp 4.20	Semi	Designtime	Dynamic	WS	Distributed
ASWComp 4.21	Automatic	Runtime	Dynamic	WS	NA
TNSComp 4.22	Automatic	Runtime	Configurable	Net	Local
SPWComp 4.23	Semi	Designtime	Dynamic	WS	Distributed

– *NA:* Not Available – *Semi:* Semi-Automatic

– *WS:* Webservices – *Net:* Network

Table 4.1: Comparison of Composition Approaches

tion, flexibility, domain and scope. The table 4.1 shows that majority of the intermediate approaches are designed for non-distributed composition and they are dynamic in nature. Almost all of the approaches target Webservice domain and they require constant human-feedback during the composition process. The presented approaches do not differentiate the time-phases, rather human-involvement during the composition process make them intermediate (semi-automatic) in nature.

Comparison to Presented Approach (Template-Based Composition): Except one4.22 intermediate approach all others are designed for Webservice domain, which poses different requirements (i.e., a larger number of services, lesser time constraints) than the network, thus, it is acceptable to have a constant user feedback during the composition process. Involvement of human during the setup time in the network will not be acceptable, which makes web service domain based approaches inappropriate to be used in the network domain. However, the flexibility of description of interface and service can still be of use.

The template-based composition approach focuses on the network domain. Hence, it requires urgency in the composition process. Unlike approaches (i.e., defined in intermediate composition related work4.14) the approach in this thesis refers intermediate nature to a split that is defined between time phases regardless of human involvement. The process at the design-time may also be carried out by an expert system without the

need of human experts.

The approach "TNSComp" also deals with network and uses the both design and run-time for composition, however, the constructed templates are directly associated with implementations of protocols. The provided flexibility is more about the selection of different Services (protocol stack), which are already composed beforehand and only require some configuration to execute. Whereas the template-based composition does not deal with the implementation of protocol until later at the run-time, which makes it highly flexible to changing inputs like application requirements and network constraints. In the presented paper [FS13], a flowchart is described, which only talks about selection and configuration of service rather than its composition. The approach is nearer to SILO [VWR+07], where meta-layers are predefined.

Chapter 5

Requirements-aware, Template-Based Protocol Graphs

This chapter describes the template based composition approach. A composition can be carried out at different time phases (e.g., design-time, deployment and run-time - as described in 1.1). The presented approach rather than using a single time-phase uses all three of them to provide the desired flexibility and to reduce the required setup time for generating the PGs. In the following text, the main idea of the template-based composition is described followed by the details on components (e.g., Template Description Language, Domain-based Policies) and the actions (e.g., Selection of template(s), Filling placeholders, BBs connection) required to generate executable protocol graphs. In the end, an example is provided to show that how the approach can generate the PGs for Secure-TCP like protocol.

After the generation of PGs if the composition process originates more than single protocol graph, all of them are executable and also satisfy the provided functional requirements. They only differ regarding QoS parameters (e.g., delay: time needs for execution of the included BBs). It will be desirable to select a PG that is nearest to the expectation of the application about QoS. The selection of an optimal PG is outside the scope of PG composition, hence the details of PG-Selection is beyond the scope of this work. The selection of a most suitable protocol graph is carried out by using qualitative properties and by employing the MCDA approaches such as MAUT and AHP, the further detail can be found in [Saa80] [Saa08] [KR76] [EG03] [Mun95].

5.1 Template Based Composition Approach

To create a requirements-based protocol graph out of the given functionalities, it is necessary to define data and control flow of the selected functionalities. A control flow is defined by the execution order while data flow defines how and where data will be transferred next. Template-based composition is a partial-run-time approach where the ordering of functionalities and their connections are defined at the design-time. The idea was also used and demonstrated in the cross-layer composition with the series of publications [MSBee12] [SJMM11] [MSH10] which was the part of G-Lab Deep project [1].

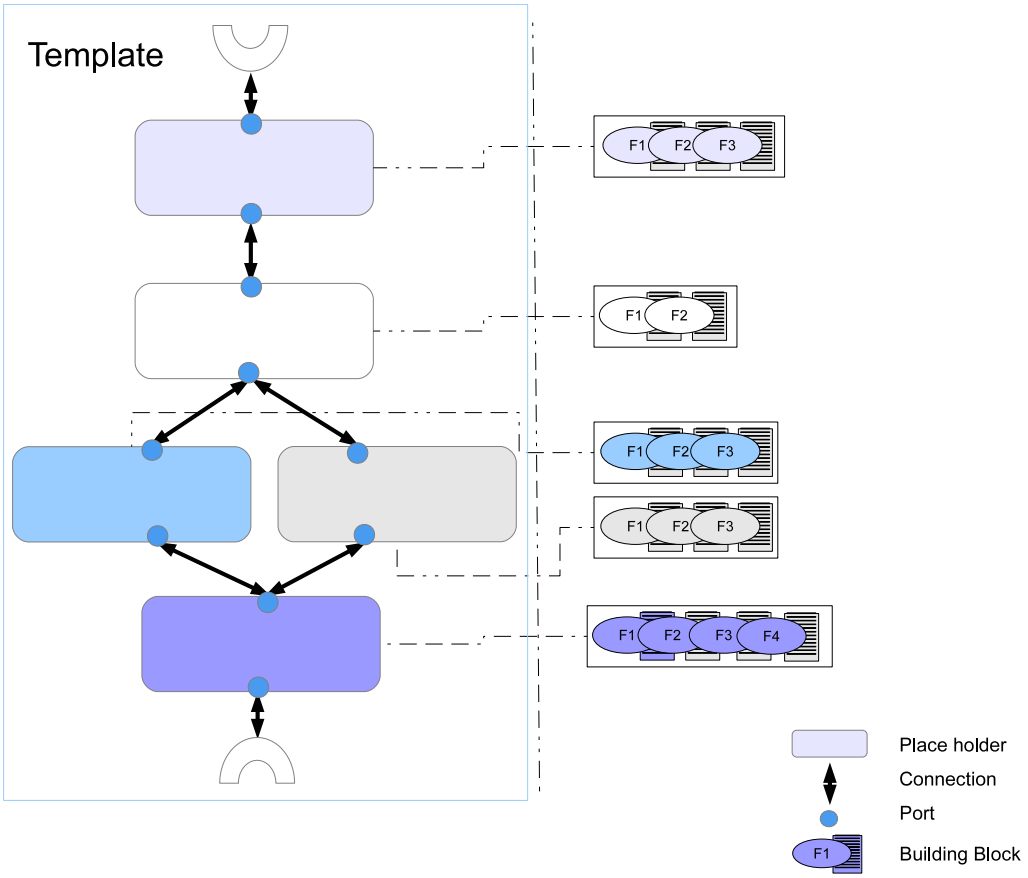


Figure 5.1: Template

The basic idea of the approach [SKM12] [KSMB14] [SM11] is to split functional composition process among different time-phases (i.e., design-time, deployment-time, and run-time) so that relatively inefficient activities in terms of time are performed at the design-time and potentially less time-consuming activities are performed at the run-time.

[1] http://www.g-lab-deep.de/index.html

In this case, time-consuming activities are the selection and the placement of functionality, but not the actual building blocks (i.e., selection of encryption and compression functionality but not their implementations/BBs such as AES256 or Blowfish256). And then placing then in an appropriate order (i.e., encryption is placed after compression) in addition to connecting them so that they can interact with each other. The template based composition approach utilizes a devised abstraction of placeholders instead of using actual functionality as shown in fig.5.1. A placeholder can be occupied by different BBs, if a BB ports match with a pace-holder ports.

Placeholder is an abstraction that encapsulates the details of used implementation (BB) for provided functionality. Place-holders are connected via ports these are only mean for communicating the encapsulated BB(s). Functionality can have multiple effects and they are distributed over the connected ports. Effects are further classified by offered (i.e., provided by the port) and required (i.e., accepted by the port) effects as shown in fig. 5.2 (a). The benefit of having ports and well-defined effects on them is that any building block that is fulfilling the given connections requirements can fill a place-holder. Hence, it is not dependent on a certain building block but rather dependent on the functionality of a building block.

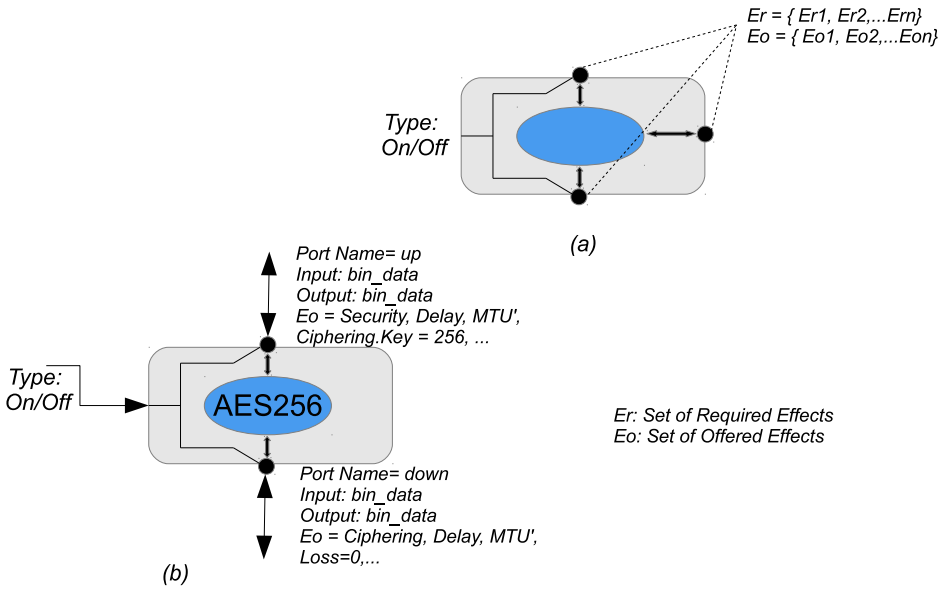


Figure 5.2: Place-Holder [SKM12]

A mechanism can have numerous implementations and may differ in terms of defined ports (i.e., it implies same covered and required effects on those ports). Any implementation which also has same ports as described in a placeholder can be a suitable match

to fill that placeholder. Figure 5.2 (b) shows the example of ciphering as an effect and encryption as a mechanism and the AES256 as a selected implementation.

A placeholder also contains optional ports that cover general interfaces like management, administration, and monitoring. Those ports are active only when a selected BB also provides that data, hence, they are not considered in the BBs selection process. The current implementation only supports port for error-notification, however, further administrative ports can be introduced as the concept matures and demand grows.

A template can be **designed** by different entities such as at design-time by an expert with domain specific knowledge. In first prototype this is the only considered option. However, it may also be possible to automatize the process by a software module (e.g., dynamic functional composition method), or having an Artificial Intelligence (AI) learning process, which keeps track of requirements and offerings and creates a template for the often used requirements3.2.1.

5.1.1 Template Description Language

A template description needs to cover the effects it provides. It also needs to describe placeholders for the possible BBs to fill them and the connections among those placeholders. Also, it is expected to provide information regarding QoS parameters. The information in the language is divided into separate sections. The covered effects are needed for the selection of a Template, whereas the BB selection process needs placeholders. The connections are needed for the proper placement of functionalities and QoS parameters are required to select the most suitable PG out of many. A template is programmatically expressed in XML language. XML is chosen for its extensibility and human readable form. The figure 5.3 shows the schema of the description language.

The template description is split into four main parts so-called Domains, Placeholders, Connections and CoveredEffects as shown in fig.5.3. Where, the *Domains* section describes the types of domains that are covered by a template, examples of domains are telephony, video streaming, video conferencing, data transmission, image transmission, file transmission, on-line banking, etc. The *Placeholders* section incorporates the existing functionalities in a template that is further sub-divided into individual placeholder and its ports. The *Connections* section of the language deals with the ordering and the connections of its placeholders. The "QoS_Parameters" section covers the QoS values of a template such as overall expected delay or encryption strength if any provided.

To give an idea how an actual template looks like, the figure 5.4 shows an example

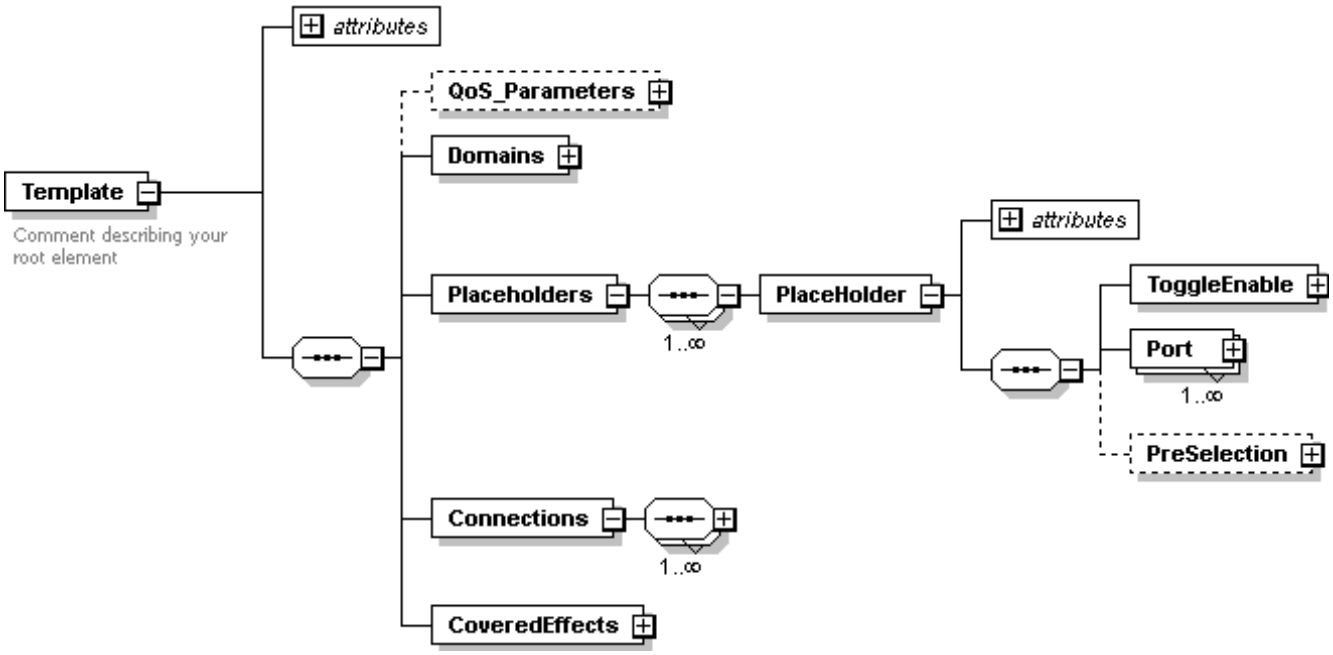


Figure 5.3: Template Description Language Schema

of a template with two functionalities namely "Compression" and "Encryption" This template consists of four placeholders, two of them are special ones, one of them provides a connection to an application and another one to a network. The other two provide the functionalities like Ciphering by encrypting data and Data Reduction by compressing data respectively. The "isToggle" attribute of the "CoveredEffect" tag informs the template selection process if a covered effect in a template can be toggled. Toggling of a covered effects helps in the sense that if a template has more functionality than requested by an application then extra ones can be turned off. Actual toggling takes place in placeholders, that is why placeholder also has an attribute "isToggle" it is a crucial property for the short term adaptation. The either of the covered effects, shown in fig. 5.4, can be excluded by turning on or off the placeholders (i.e., compression, encryption) without creating any malfunction in a PG processing.

Further examples of templates are listed in the appendices A.

Placehoders: The template example shows a compression placeholder which accepts plain data as an input (i.e., requiredeffect) at the port "9" and gives output (i.e., offeredeffect) compressed data ("DataReduction") at the port "2". This process is reversed when data is coming from the network.

Data types are not part of the description as they are mainly used for to connect

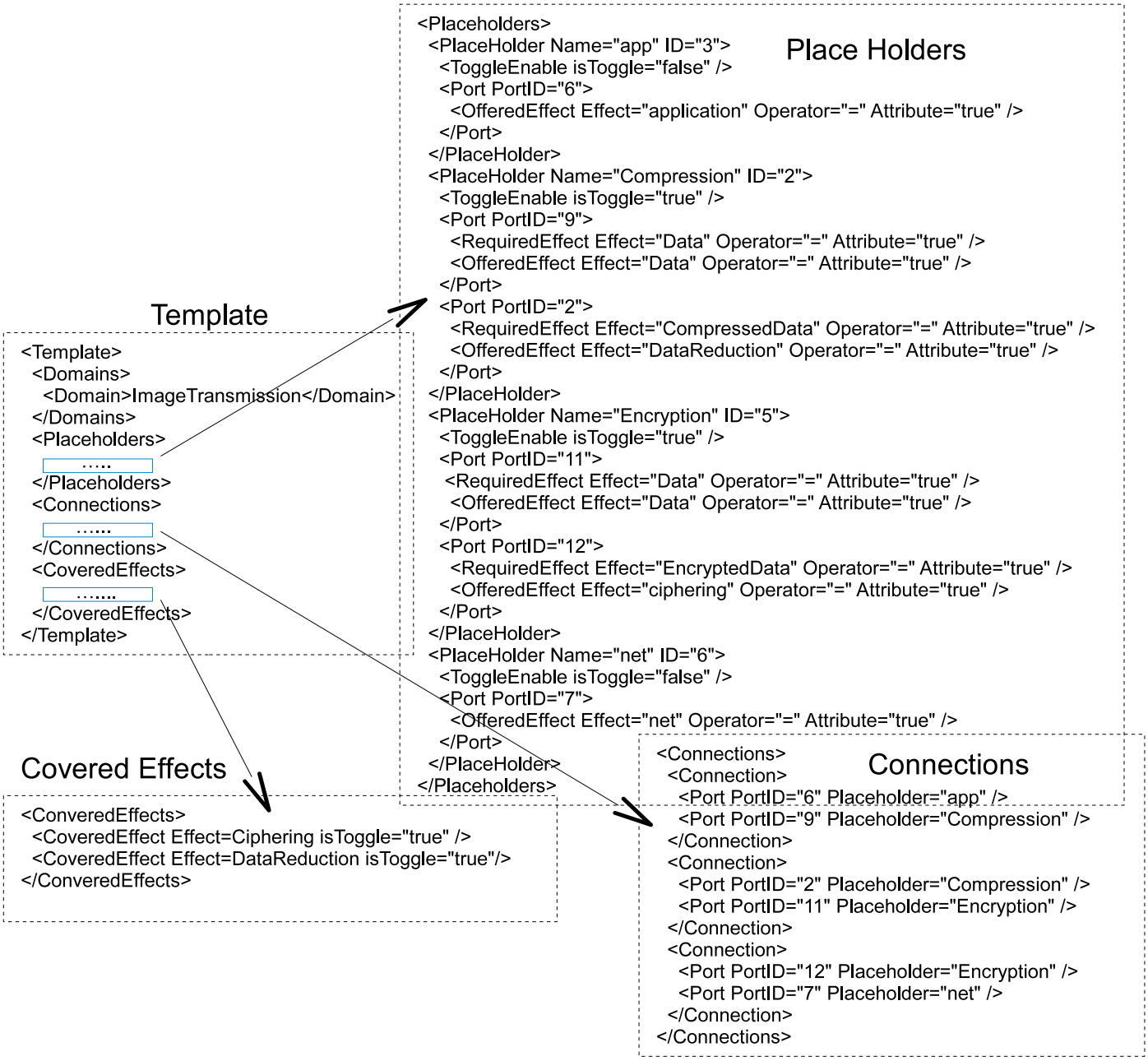


Figure 5.4: An Example of a Template

two placeholders and as connections are already part of the template description so that defining data types on ports is redundant.

Placement in a Template (Connections): The connections section in the example shows connection among four placeholders namely app, Compression, Encryption, and net. Where compression is placed before encryption, this connection example also enlightens the fact about the importance of placement of the functionalities as encryption before compression makes the least sense. The *CoveredEffects* section describes which functionalities are part of a template, which is also used for the selection of a template.

5.1.2 Domains Policies and Its Description Language

Domains Policies

A template is created for the certain **domain(s)**, the need of domains is to include the functionalities that are required by them but not requested by an application. The examples of domains are telephony, messaging, data-transmission and image transmission. One template can cover more than one domain, in other words one template can consist of functionalities that are used in multiple domains. The domains which are covered in a template are listed in the "Domains" branch of the template description5.1.1.

The functionalities like data reduction, connection management and loop avoidance as shown in figure 5.5 are of no importance for an application or a user but many of them are essential for communication such as for addressing or connection management. Hence, extra policies or requirements from an administrator or a network are needed to describe them that are referred as **domain policies** in the template-based composition.

The requirements from an application also include the domain-name such as an Image-Processing application may provide a domain name as "ImageTransmission". Later, based on a domain name, its policies will be extracted from the predefined domain policies. These policies are used for listing the functionalities, which are not part of application requirements but crucial for optimization or proper working of a communication. Many of these policies are a simple list of functionalities and others are conditional. An example of a conditional policy will be that in case of low (i.e., less than given value) bandwidth, a compression functionality will be included in the given requirements.

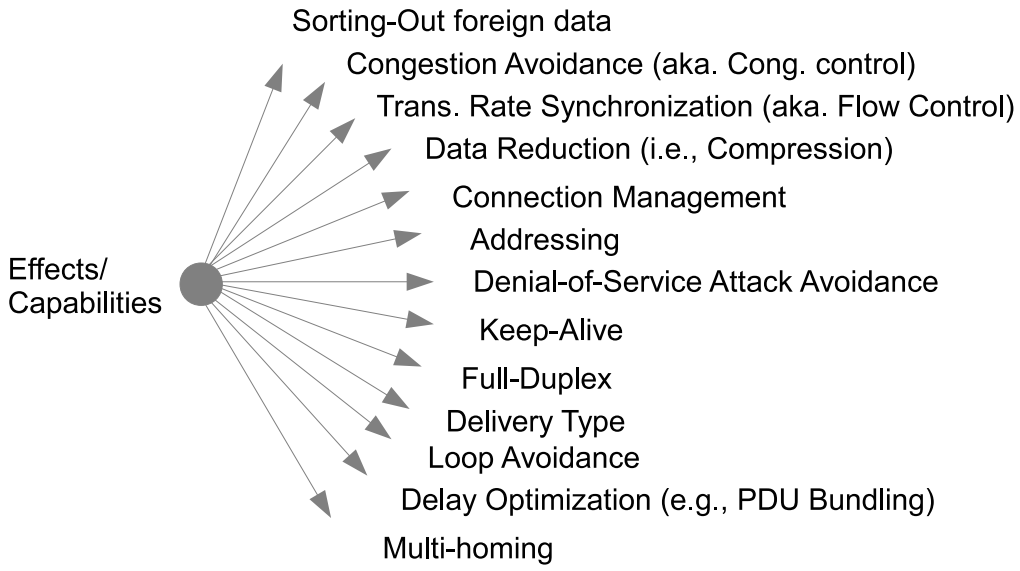


Figure 5.5: Example of Effects for Policies

Description Language for Domain Policies

A simple description language is used for listing the policies of a domain on a system. The language consists of the section called ”Domain” this is divided further into two subsections namely ”Requirement” and ” Condition”. The schema of the language is shown in figure 5.6. The language supports two kinds of inclusion of functionalities with and without condition. In the case of without condition inclusion, a name of functionality is added under the tag of ”Requirement” for a particular domain. The conditional inclusion is shown by an XML listing in 5.1, where two different domains ”DataTransmission” and ”ImageTransmission” are listed with a simple network policy. The policy in this example is conditional, the ”DataReduction” effect will be turned on only when bandwidth is below a certain limit.

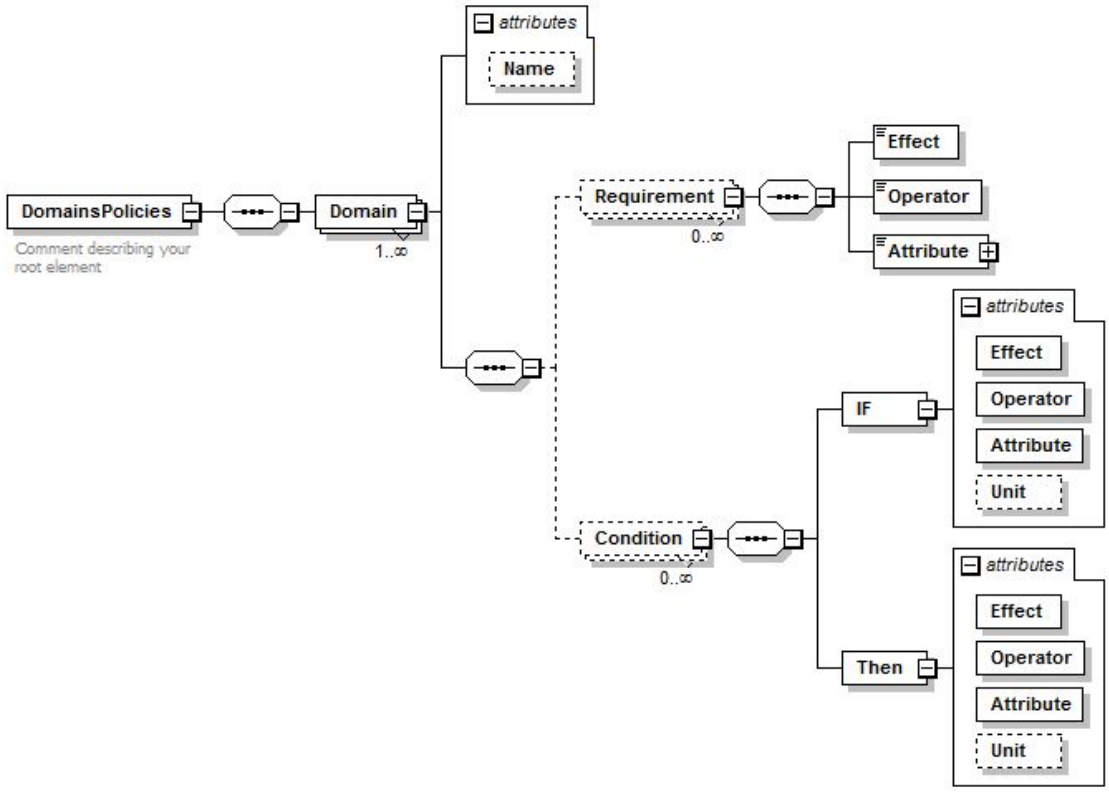


Figure 5.6: Domain Policies Schema

Listing 5.1: Domain Policies

```
<?xml version="1.0" encoding="UTF-8"?>
<DomainsPolicies xmlns:xsi="" xsi:noNamespaceSchemaLocation="/
   DomainPolicies.xsd">
        <Domain Name="DataTransmission">
        <Condition>
                <IF Effect="bandwidth" Operator="<" Attribute="2" Unit=
                   "MB"></IF>
                <Then Effect="DataReduction" Operator="=" Attribute="
                   true"></Then>
        </Condition>
        </Domain>
        <Domain Name="ImageTransmission">
        <Condition>
                <IF Effect="bandwidth" Operator="<" Attribute="1" Unit=
                   "MB"></IF>
                <Then Effect="DataReduction" Operator="=" Attribute="
                   true"></Then>
        </Condition>
```

```
        </Domain>
</DomainsPolicies>
```

5.1.3 Selection of a Template

To select a template, first of all the selection process receives a set of requirements from an application or a user. The requirements are represented by a quadruplet of effect, operator and attribute besides its weight (i.e., to specify its importance for an application) as described in 3.2.1. In figure 5.7, a set of application requirements is represented by R_p, set of QoS parameters by Q_i, the set of domain policies by R_d and a set of offered effects by O_e. There is no difference between the description of application and QoS requirements. Thus, QoS parameters cannot be differentiated from functional requirements. Template selection process knows all the QoS (i.e., denoted by Q_p) beforehand.

After receiving the application requirements, the domain is extracted and it is checked against the stored domain policies at the system. Once the domain policies are retrieved, it is merged with the application requirements ($R_p \cup R_d$) to find a matching templates as shown in figure 5.7 The quality of service (QoS) parameters are extracted (R-Q_p) from the combined requirements. QoS parameters do not play any role in a template selection, as QoS of a communication service is affected by the implementations of the selected functionality, which is not carried out during the template selection. There are two types of QoS parameters, one which is related to entire workflow such as delay and other is specific to BBs such as the strength of encryption (ciphering.key). Both of them are extracted from the merged requirements. Overall QoS of a communication service is dependent on combined QoS of individual BBs that are used in the BB selection process.

The template selection process also neglects the requirements related to BBs and these do not influence the selection of a template such as Ciphering. A requirement like Delay $<$ 2ms does not alter the template selection but a requirement for the selection of a suitable BB.

The matching process goes through all the merged requirements and checks against covered effects in a template. This selection process works separately for a single template, the advantage of this approach is that multiple processes can run in parallel without being dependent on each other.

After the selection process is executed, if more than one template satisfies the final requirements then either a single template will be selected or all the templates will be selected as described below.

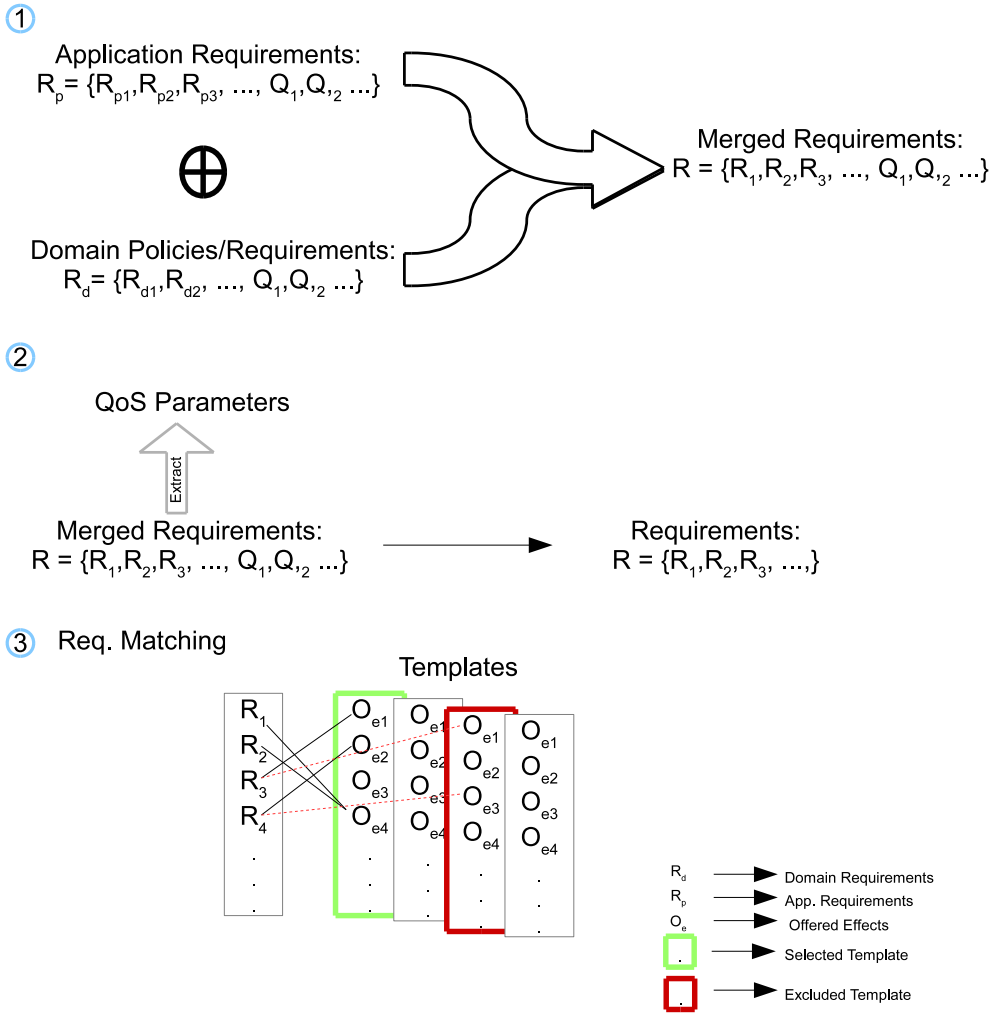


Figure 5.7: Selecting a Template

- **Exact First Match:** It is the simplest strategy to choose the first template that satisfies all the given requirements without any additional covered effects.

- **Select All:** This strategy forwards all the possible templates for the requirements to the BB selection process so that, the best possible protocol graph based on QoS parameters is selected from the available choices.

Figure 5.8 shows the flowchart of the template selection process where *select all strategy* is deployed to retrieve all the suitable templates.

Example: An example can more clearly illustrate the template selection process. An application wants to have a secure & reliable transmission of an image. Thus, the application specifies the requirements as ciphering and retransmission besides the domain

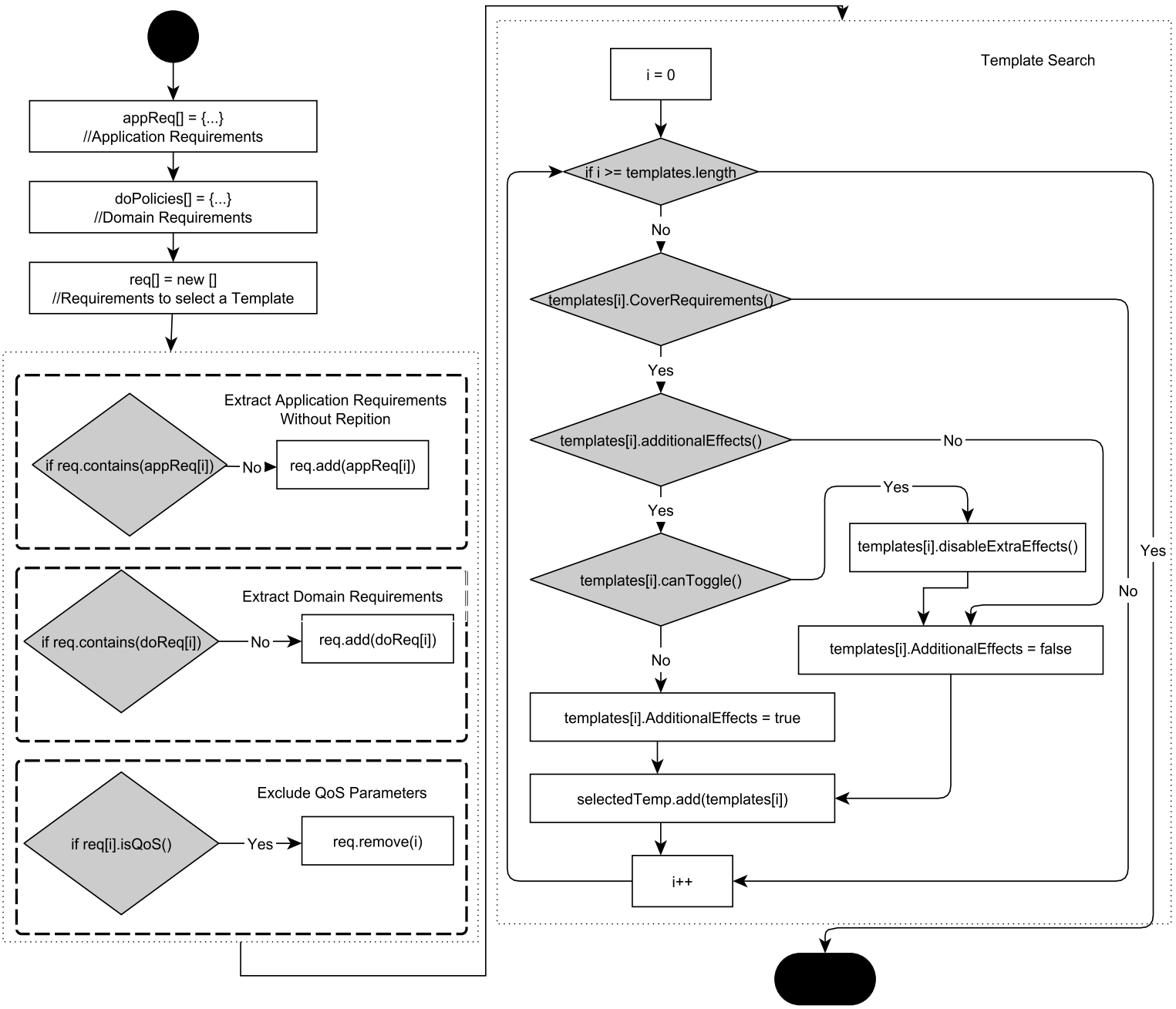


Figure 5.8: Flowchart: Template Selection

Application Req.:
(Security) Ciphering = true
(Reliability) Retransmission = true
Domain = Imagetransmission

Domain Req.:
Ciphering = true
Ciphering.Delay < 2 ms
Ciphering.Key = 256
DataReduction = true
Ordering = true
Delay < 10ms

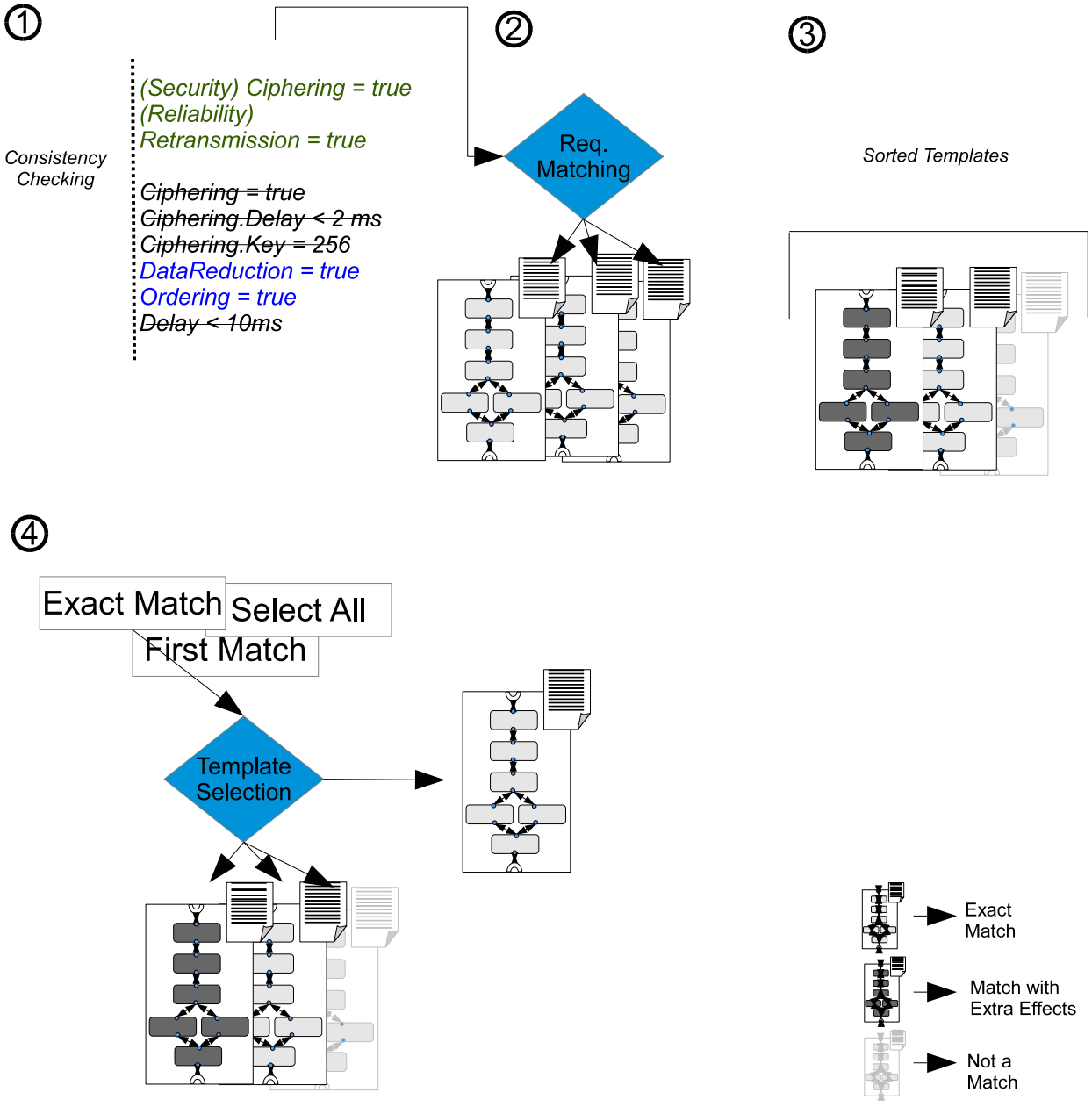


Figure 5.9: Scenario for Template Selection

of data as ImageTransmission and VideoTransmission as shown in figure 5.9. However, optimizing features like DataReduction is not specified by the application, which can be further included in the domain requirements. At the first step, consistency of the requirements will be checked it is to exclude any repetition (Requirements Union). QoS parameters are excluded related to functionality or entire protocol graph. As shown in the figure 5.9, the "Ciphering" is included only once besides, the QoS parameters such as "Ciphering. Delay", "Ciphering.Key" and "Delay" are not part of the requirements for the template selection.

In the second step, the process searches for the templates having the covered effects to satisfy the incoming requirements. In this case, any template provides the functionality of Encryption, Retransmission, Ordering and Data Reduction will be sorted out.

Once the templates are sorted, in the 4th step an algorithm is executed to select the template(s). In this scenario, exact first match strategy is deployed. Thus, the only template will be selected which covers only the given requirements. However, if there is no exact match then next best match will be selected with additional effects and if there are more than one exact match then the first exact match is chosen.

5.1.4 Finding Suitable BBs for Template's Placeholders

To fill a template, every placeholder that can not be switched off must have at least one BB which can be fit in it so that the requested service can be satisfied. However, there can be more than one BBs which can be fit in a placeholder these are called "suitable" building blocks as shown in fig. 5.10.

Selection of Suitable Building Blocks: Suitable BBs are chosen by matching ports and its effects of building blocks with the provided placeholder ports. The selection also based on given application requirements for the functionality as shown in fig. 5.11.

For matching application requirements with the building block offerings, each effect must be uniquely identified as presented in these papers [SM12] [SDS+09] [KRS+10].

A port of a placeholder and a BB provide a set of offered effects (OF) and required effects (RE). Hence, to be sure if a BB can be fit into a placeholder, a BB must provide the matching ports to the placeholder ports as shown in figure 5.11.

To match BB's port to placeholder's port, the set of OF and RE of them are compared. If a match is found, the process moves on to the next ports until all the remaining ports of the placeholder can be connected to the ports of the BB. Once, it is known that all the ports of a placeholder can be occupied by a BB, a reference to this BB will be added

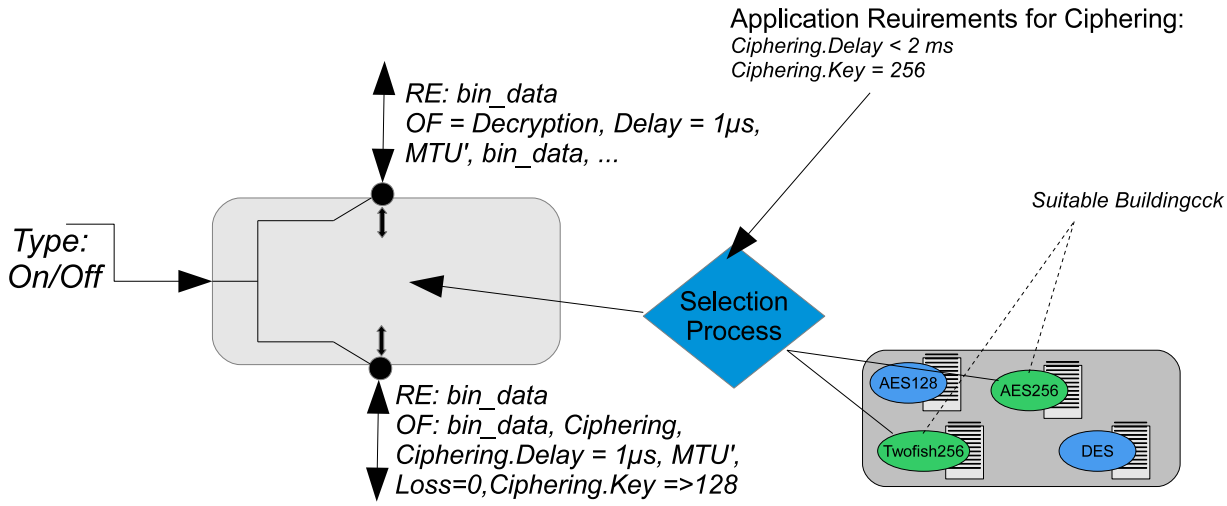


Figure 5.10: Filling a Placeholder [SKM12]

as a suitable BB for the placeholder. If a BB's port offers more OF then it is needed such as additional formatting of data, it is still considered a matching port if rest of the effects can be neglected in the protocol graph execution. The figure 5.11 shows two BBs, where BB-AES256 matches both of placeholder ports and BB-3DES only covers the single port. Thus, BB-AES256 is a suitable BB but not BB-3DES. The figure also depicts the condition of the additional effects on a BB's port where BB-3DES offers an additional effect of "quality" on Port-1 and, still this port is considered a matching port as the effect does not change the data.

The figure 5.11 also shows a case where a BB offers the ports which are a not needed by a placeholder, the port 3 of BB-AES256 remains unconnected and will be neglected during the execution of a protocol graph.

The process of a BB selection is described in the flowchart 5.12, where all the available building blocks in the BBPool are looped through to match against the placeholders in a template. A suitable BB must cover all the ports of a placeholder except optional (e.g., data-logging, monitoring, management) ones. In the figure 5.12, the IsBBMatch() function is responsible for going through all the ports and once a port is found a mapping between the placeholder port and the BB port is added. Ports don't share the exact name or description, the only common thing is set of covered and offered effects on a port. If a BB offers more than single port for a port in the placeholder then, the first port will be selected as the loop breaks as soon as a suitable port is found. To check the effects, first the process goes through all the required effects on a port if all the effects are matched then the process checks for the offered effects. In case, any single effect is not matched

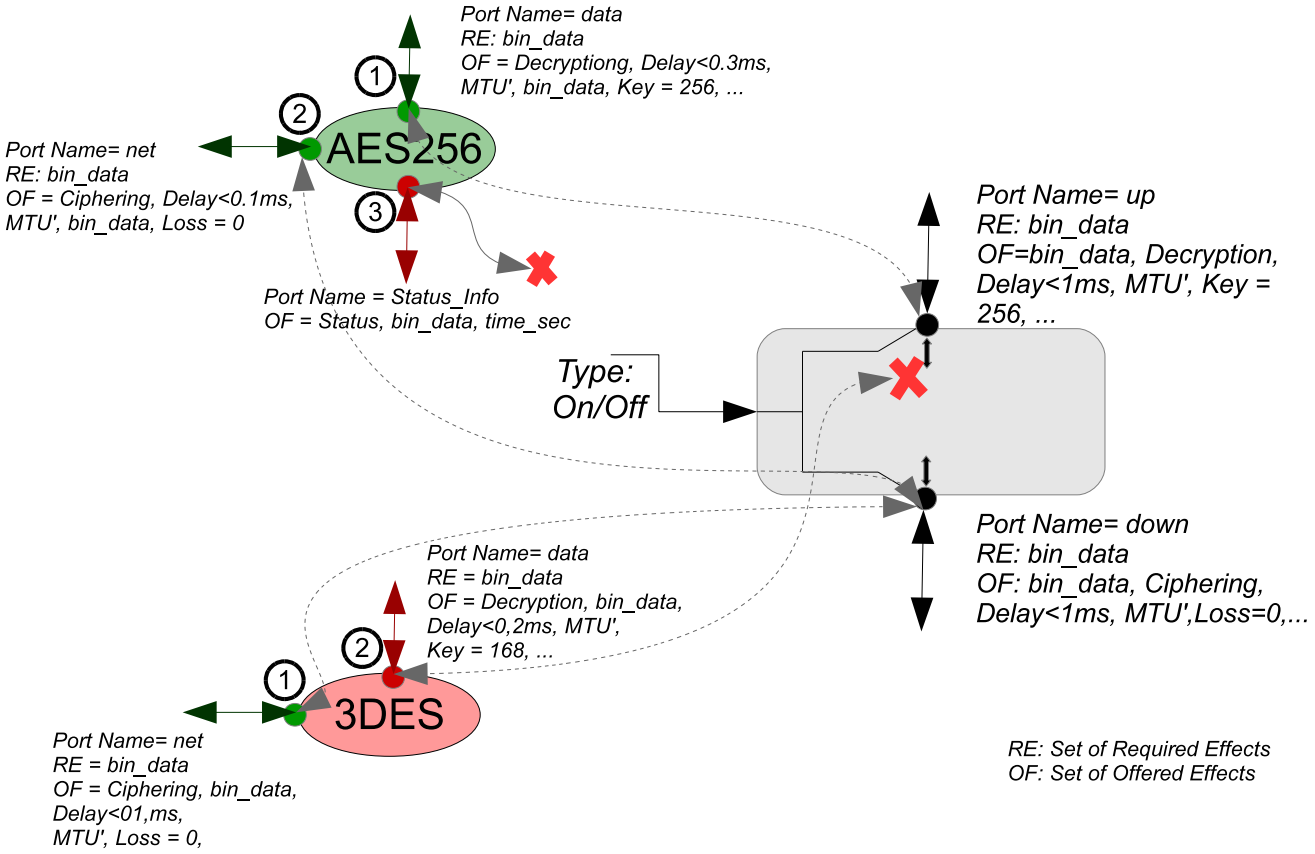


Figure 5.11: Ports Matching

the port will not be considered as a matching port.

The selection of suitable BBs for the placeholders can be performed beforehand such as at the deployment-time or as soon as a new BB is introduced into the repository. Thus, when a BB is introduced on a system, a background selection process goes through all the placeholders of available templates and in the case it matches to a certain placeholder, a reference is added in that placeholder.

The pre-selection strategy fastens the connection process. However, this does not take any requirements or constraints into account. Let's assume a placeholder, which can be filled with various kind of encryption strengths (128, 256, etc.), application requirement specifically asks for 256 key strength. If a placeholder is filled beforehand, it will have all the possible BBs with different key strengths. However, it is still possible to preclude BBs on QoS parameters (e.g., key-strength, delay) for this specific placeholder (Ciphering). The QoS based selection of BBs requires more processing time, if building blocks have

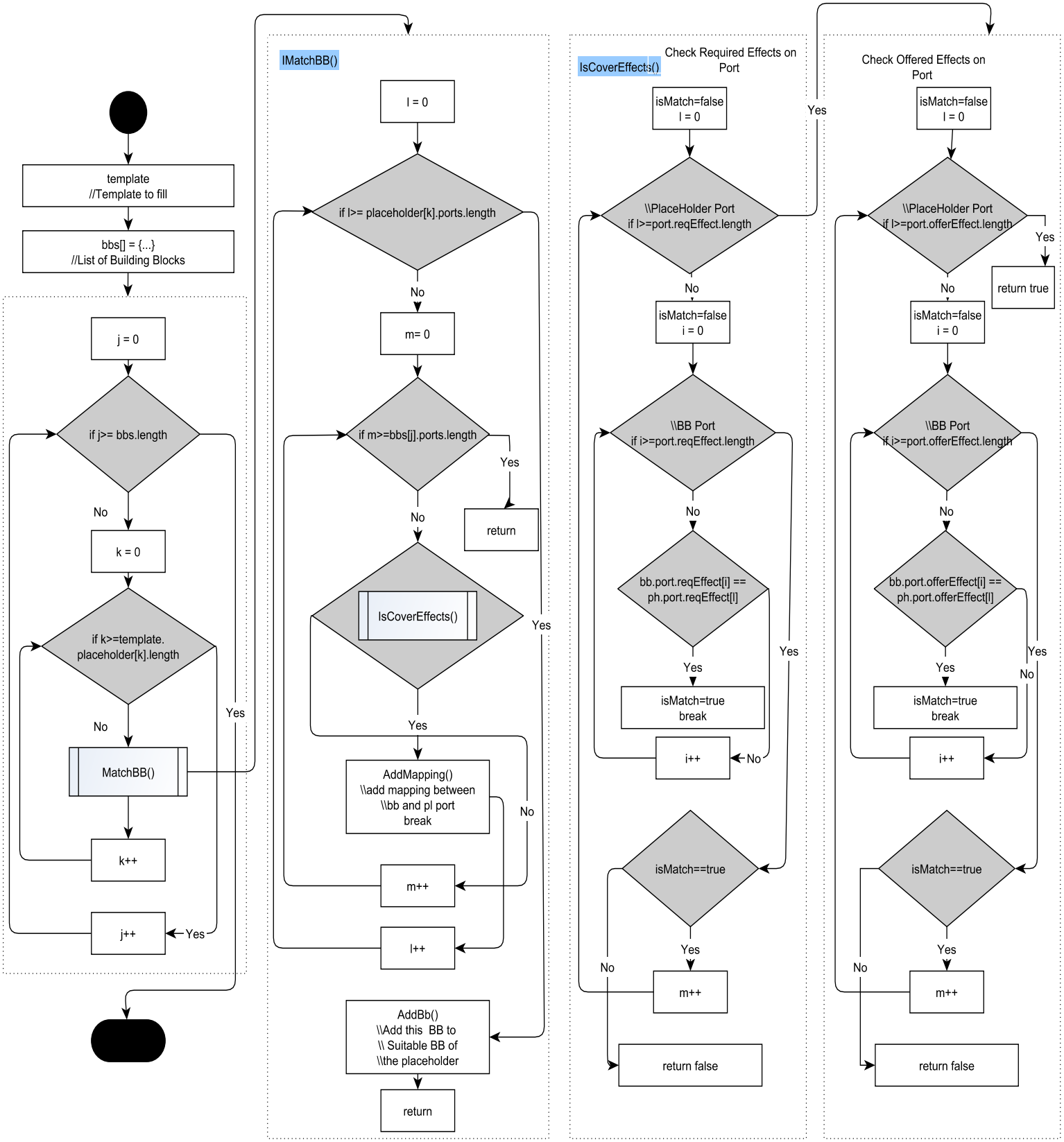


Figure 5.12: Flowchart: Filling the Placeholders of a Template

multiple selection criteria such as delay, throughput, and cost then process will use Multi-Criteria Decision Making (MCDM) methods to have an appropriate selection.

5.1.5 Protocol Graph(s) Construction

Once the placeholders of a template are associated with actual software modules (BBs), the final stage of the composition process is to generate executable protocol graph(s). The number of protocol graphs is directly related to the number of possible BBs in each placeholder.

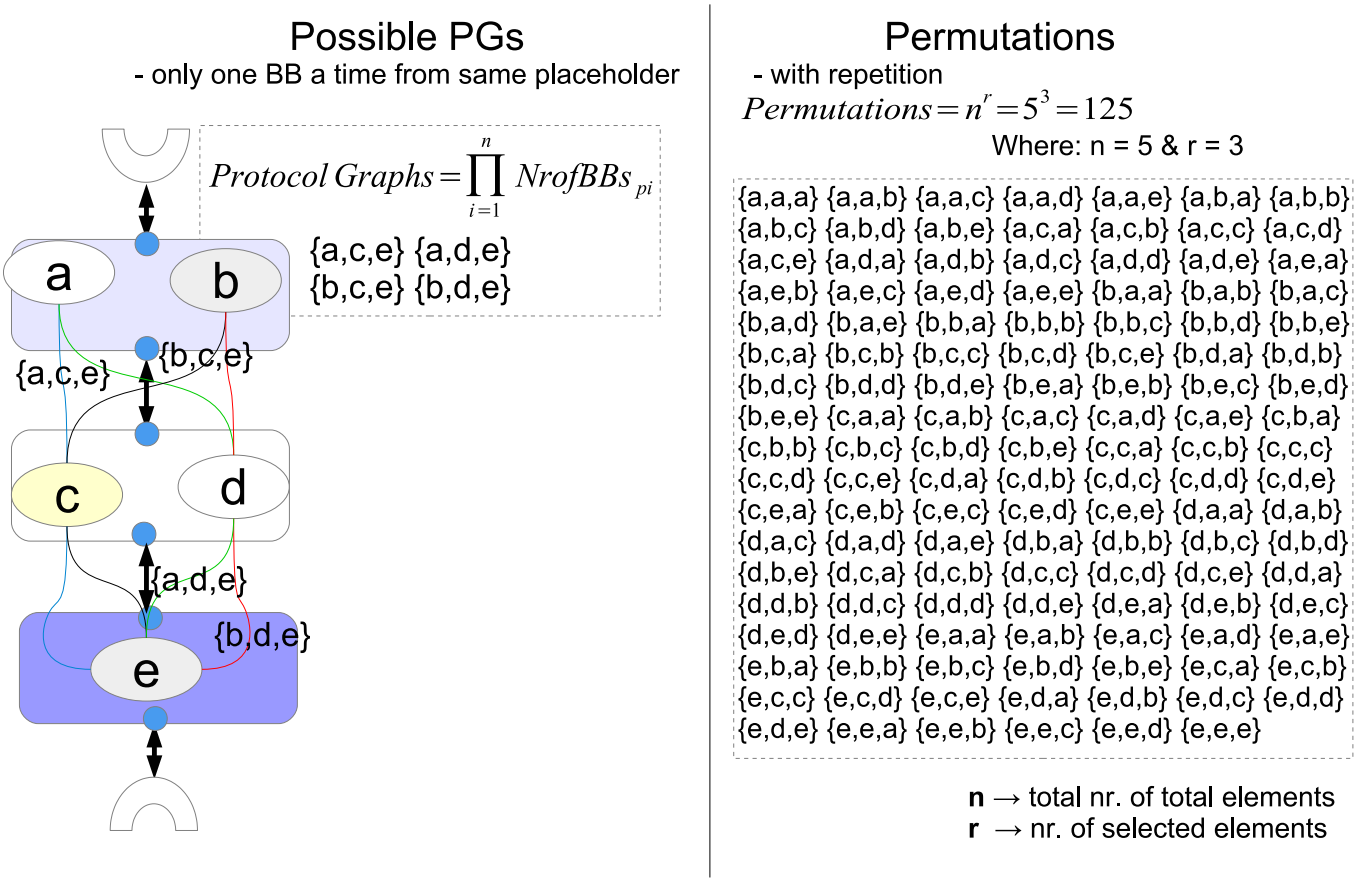


Figure 5.13: Comparison Between Possible PGs in Template-Based Composition & Simple Approach

The trivial composition approach (described in 1.2) assumes that any BB can be combined to get all possibilities. However in a template, not more than one BB can be used from the same placeholder for a protocol graph as every BB has an order that is provided by the placeholder. The comparison between two approaches is shown in figure

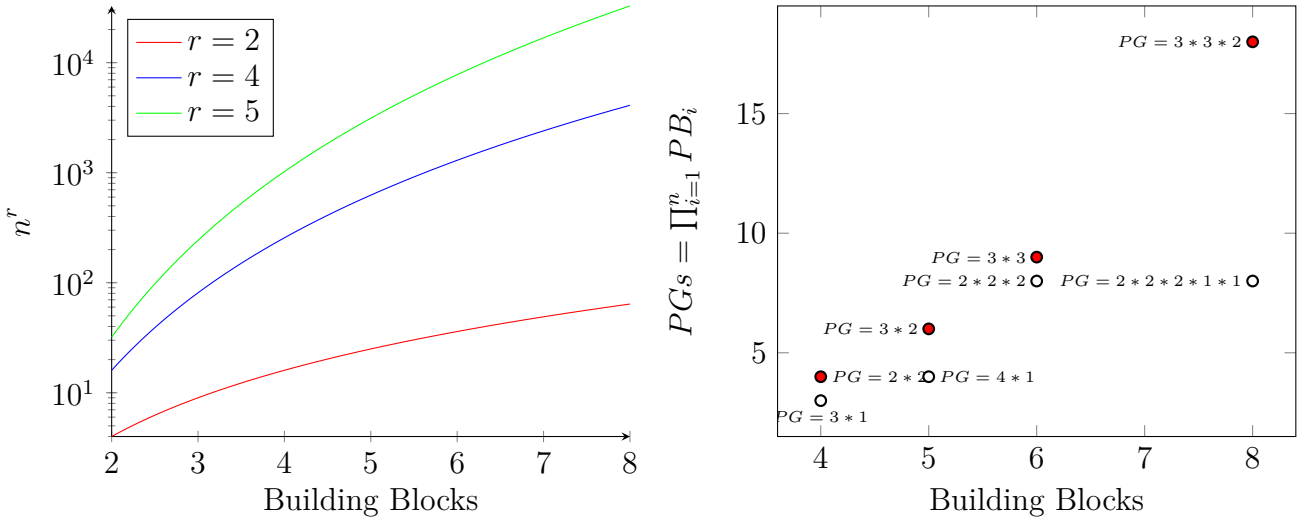
5.13, where maximum possible permutations for five (5) BBs with three (3) are chosen at a time, with repetition, and order is important are ”125” for the trivial approach.

In the template-based approach, if a placeholder has more than single suitable BB then only one can be selected at a time as shown in figure 5.13 where both ”a” and ”b” can not be part of same protocol graph.

The possible number of protocol graphs in a template is calculated with the product of all BBs as shown below.
$PossiblePGs = \prod_{i=1}^{n} PB_i$
Where PB_i is number of BBs in the placeholder-i. By given formula, it can be calculated for the given scenario of the figure 5.13:
$PossiblePGs = 2 * 2 * 1$
$PossiblePGs = 4$
It is also important to note that, a repetition is a possibility in a template, in case two different placeholders have same BB. The most common scenario will be encryption of encrypted data to have further security, in this case, both placeholders will have same BBs but at different positions in a protocol graph.

The figure 5.14 shows growth-comparison between Template-based composition and the described trivial approach when the numbers of BBs are increased. The trivial approach gives an ascending graph as it can be seen figure 5.14(a), the graph uses 2, 4, and 5 number of BBs from the total available pool for a single PG. It can be seen the number of permutations highly depends on the combination size and total available BBs.

Whereas the number of generated PGs in Template-Based composition is not dependent on the total number of building blocks but rather on the number of placeholders and how many BBs each of those placeholder holds. As it can be seen from the figure 5.14 (b), the same number of BBs and placeholders can generate the different number of protocol graphs such as 5 BBs in two placeholders can generate 4 and 6 protocol graphs depending on the distribution of BBs among the placeholders in a template. The graph also shows the maximum possible PGs which is shown by red dots for the given number of BBs, whereas the minimum number of PG will be 1 at any point. The number of maximum possible PGs is created by distributing the BBs across a different number of placeholders such as if BBs are 5 then only if these BBs are distributed into 2 placeholders will result in maximum PGs as shown by the following calculation.
$PossiblePGs = 2 * 2 * 1 = 4 \quad where \quad BBs = 5, \quad Placeholders = 3$
$PossiblePGs = 1 * 1 * 1 * 2 = 2 \quad where \quad BBs = 5, \quad Placeholders = 4$
$PossiblePGs = 3 * 2 = 6 \quad where \quad BBs = 5, \quad Placeholders = 2$



(a) Simple Approach (b) Template-Based

Figure 5.14: Comparison of Possible PGs Growth

How protocol graphs are generated in the approach is explained by an example for the domain of "Imagetransmission" as shown in the figure 5.15. At first (Step 1), an application asks for encrypted and loss-free data transmission by specifying the requirements. Once the requirements are received (Step 2), application's domain is retrieved to check the domain policies. Here, the domain has a conditional policy which states if bandwidth is less than or equal to one megabyte (1 MB) then a data-reduction (compression) functionality will be included in the PG. The next step (Step 3) is to check network parameters to decide whether to include data reduction or not. The example network offers the bandwidth of "1 Mb" and "0" data loss ratio, which implies that data reduction must be part of the protocol graph. Now (Step 4), the selection process goes through template repository to find suitable template(s) if there are more than one suitable template than the first exact match will be selected. In this example, a single template is selected. The template consists of three placeholders namely Compression, Encryption and Retransmission. These placeholders have pre-filled references of the suitable BBs, where P1 holds a reference to "ImageComp75" BB, P2 has two references of "AES256" & "Twofish256" and the P3 holds a single reference to "Retransmission" BB.

Next, the possible number of protocol graphs is calculated by the product of available BBs in the placeholders that are two (2) in this scenario. The PG generator will iterate through 2 times to create two protocol graphs, as it can be seen from the figure 5.15 in step 5. The only difference between two PGs is different encryption mechanism with a change in delay. It is also important to note that both of these protocol graphs are

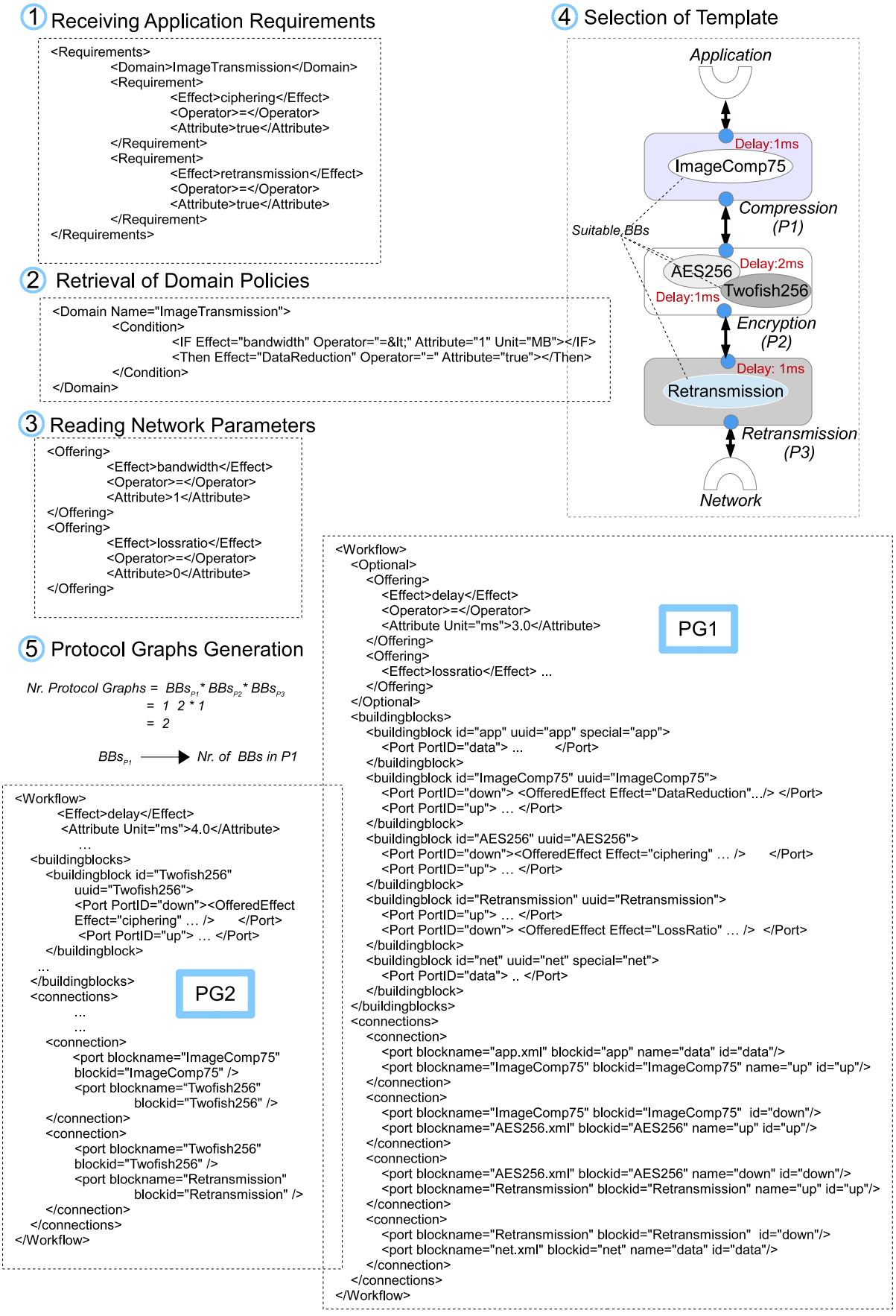


Figure 5.15: Example: Protocol Graphs Generation

executable. The generated XML files of the protocol graphs are listed in Appendix A.

5.2 Putting It All-Together - Secure TCP Example

To create a TCP-like protocol graph. First, we need to identify the needed functionalities to satisfy the requirements before creating a template. The identified functionalities also need classification of application requirements and domain policies. An application will not be interested in a connection management rather in fastest possible transmission mode. The following functionalities are included to fulfill the TCP-like protocol requirements.

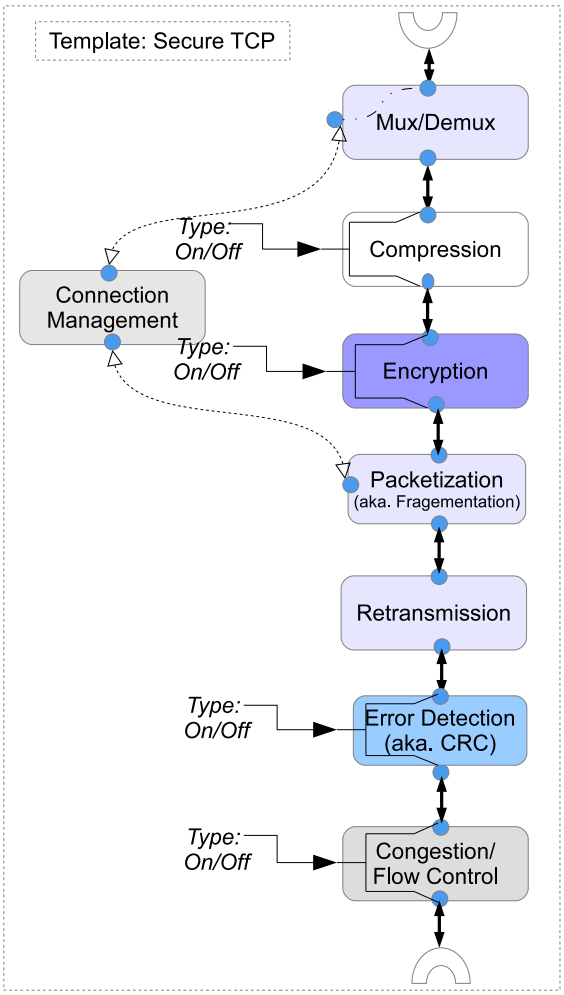


Figure 5.16: Secure-TCP Example

- **Mux/Demux:** On a single system, multiple applications use the same protocol graph to communicate. This functionality provides the added information (e.g., port number) to distinguish different data streams (Multiplexing). It also sorts out (De-Multiplexing), once the data arrives at the desired destination.

- **Error Detection:** This functionality controls data correctness by detecting any error caused by application/network malfunction or by an external threat. TCP uses Cyclic Redundancy Check (CRC) to perform this task.

- **Packetization:** This functionality is deployed to avoid large chunk of data that can not be handled by a network path. It is performed by dividing the data into sizable packets to smallest Maximum Transmission Unit (MTU) acceptable on a network path. Besides network, functionalities can also be a reason for smaller MTU in case they can handle the only limited amount of data at once such as underlying technology ATM, Ethernet, Point-to-Point Protocol (PPP).

- **Retransmission:** Retransmission avoids the data loss and ensures the reception of the delivered data in a communication. This functionality resends the packets after a certain interval if an acknowledgment for them is not received back. TCP uses Automatic repeat request (ARQ) for this purpose.

- **Congestion/Flow Control:** TCP uses these functionalities for controlling the amount of data flow. Flow control regulates the amount of transmitting data so that a receiving end can properly handle it. Whereas congestion control tries to manage the well being of overall network. It controls the amount of data transferred at once.

- **Connection Management:** This functionality provides different procedures such as connection establishment (TCP: Three-way handshake), connection release and keeping connection alive.

- **Extra Functionalities** (Not Part of Current TCP Implementation)

  - **Compression:** Compression reduces the data size in order to save valuable bandwidth and to shorten the transmission time.

  - **Encryption:** This functionality protects the data by encoding it in a ciphered text.

An application requests for a ciphered and reliable transmission (e.g., encryption, retransmission, error detection). The *domain policies cover the rest* of the needed effects such as Congestion/Flow Control, Multiplexing and compression.

Figure 5.16 shows the covered functionalities and theirs placement. The covered functionalities in a template are placed in an executable order. In this example as shown in the figure 5.16, control mechanisms and CRC are placed nearer to the network preceded by retransmission as a packet should to be checked for error first and thrown away in case of error before it is delivered further. Compression is placed before encryption for optimal function. Encryption cannot be placed after packetization as packet number & size are needed by the functionalities below. Connection management is placed parallel to the other functionalities, as it is not needed during the entire communication time. The complete template is listed in the appendix A.3.1.

The template also provides some entirely independent functionalities such as compression, encryption and error detection. The turning off or on of those functionalities does not have any impact on any other functionality. If the application does not request ciphering, encryption is turned off without any malfunction or effect on the other included mechanisms.

The implementation of this TCP-like example contains a description of a Template, BBs description, domain policies for SecureTCP domain and the generated PGs. As the implementation of BBs is out of scope for a composition process, and its goal is to generate one or more executable PGs. In the implementation of the current example, every placeholder holds only single suitable building block. The BBs are listed in the A.5. The protocol graph generation is described in the section 5.1.5 and the resultant protocol graph is listed in the appendix A.4.1.

5.3 Conclusion of Template-Based Composition

This section describes conclusive detail on how the approach achieves the desired flexibility as well as what are the aspects of the approach which counts towards the reduction in required setup-time. However, the performance evaluation of various processes of the approach is carried out in 6.

Flexibility and Template-Based Approach

The following text discusses the flexibility in the template-based composition.

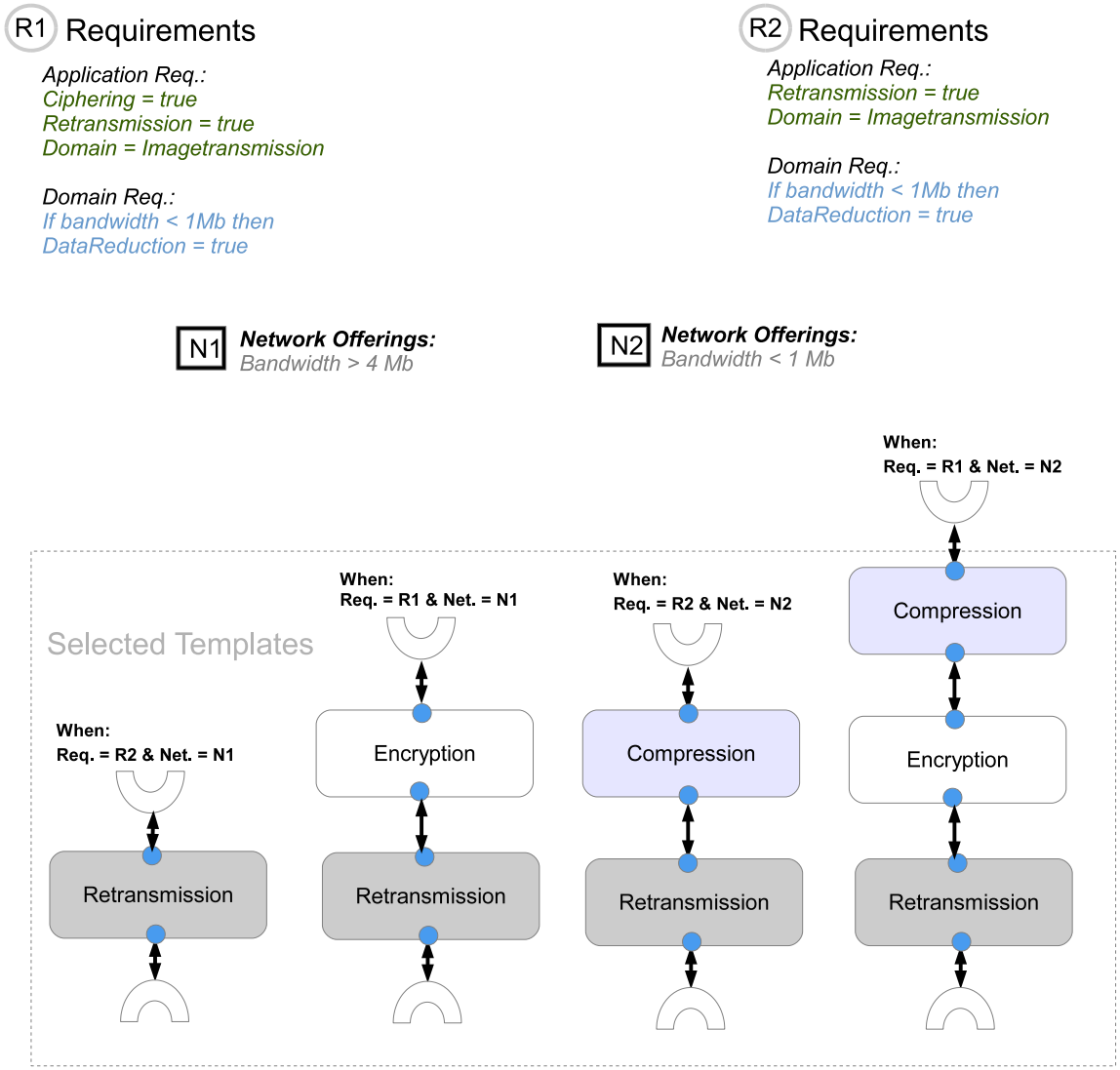


Figure 5.17: Requirements and Short-Term Flexibility

Short-Term Flexibility:

The approach uses the run-time for selection of appropriate template and implementation of functionality (BBs) to generate PG(s). The late binding of the actual implementation (BB) fosters the short-term flexibility. Whereas the maximum available information at the run-time helps to generate PGs which are aligned to the application requirements & the network conditions.

Figure 5.17 shows a scenario where two different kinds of application requirements and network offerings are given. The domain policies remain same in both cases. The requirements 1 (R1) consists of ciphering, retransmission, and the requirements 2 (R2) has a single requirement of retransmission. R1 & R2 belong to a same domain ("Image-

transmission") and share the same domain policy. It states that in the case of offered bandwidth is less than 1 Mb the size of the data should be reduced by compressing it. The system offers two kinds of network connections one with bandwidth less than 1 Mb and other with more than 4 Mb. Depending on the incoming requirements and the selected network connection different a PG will be generated. The figure 5.17 shows four different possible combinations where R1 & R2 ->Requirement 1 & 2, and N1 & N2 ->Network offerings 1 & 2 respectively.

Long-Term Flexibility:

The approach supports the deployment of a new template or a new building block. Moreover, newly deployed functionality can be immediately used, which in the long-run provides the long-term flexibility in a network. This phenomenon is further elaborated in the following scenario.

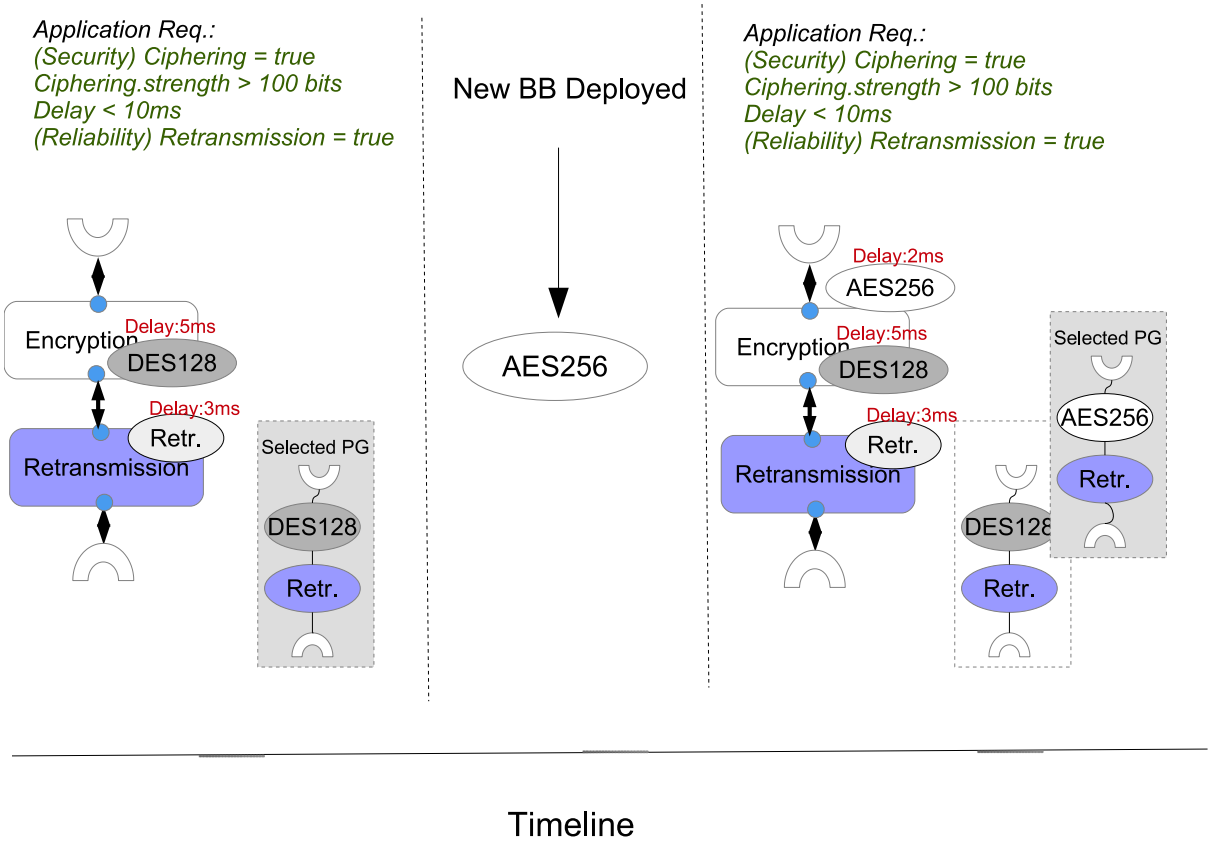


Figure 5.18: Requirements and Long-Term Flexibility

The figure 5.18 shows a scenario where an application asks for security and reliability with the QoS parameters like encryption key more than 100 bits and delay less than 10 ms. The figure also shows a time-line (left-to-right). When the requirements first

arrive the BB pool has only one suitable BB for each Encryption and Retransmission. Thus, only single PG is generated where DES128 provides 128 bit encryption key. Here, encryption needs 5 ms for the execution and retransmission needs 3 ms for the execution. Hence, overall delay remains less than 10 ms as required. The next, a new building block is deployed to the system. And it is instantly made available to be used by going through all the templates and their placeholders to check for suitability.

When the same requirements are presented to the system again it generates two PGs with two different encryption mechanisms namely AES256 and DES128. In this case the selected PG is with AES256-BB as it provides the better encryption and lesser delay (5 ms) based on QoS requirements.

Required Setup-Time and Template Based Composition

The split of functional composition among different time phases in the template based approach makes it possible to perform the tasks (i.e., which increase the required setup time) like placement and connections at the design-time to reduce the required setup time. Experts can be involved in an optimal placement of functionalities in the design-time placement that is not pragmatic during the run-time.

The connections are defined at design-time. The experts validate the connections if au automatic composition has been used to create them. The connections section of the template description language is responsible for the connections and placement as shown in fig.5.19(a). The ports of a placeholder are connected with another placeholder(s) ports, which are later mapped to selected building block ports.

How placement is looks like in the template based approach is shown in the fig.5.19(a). It shows a part of the schema (i.e., described in 5.1.1) of the template description language. The connections section of the language deals with the placement of functionality besides connections among them that is further explained by an example given in fig.5.19 (b) and (c). Part (b) gives a pictorial representation of placement and connections of placeholders. Part(c) shows how does it look like in an XML-format.

The required setup time can be further reduced by pre-selection of building blocks in the approach. The idea behind is to reduce the number of choices by selecting the suitable building blocks for the placeholders in a template. The process of pre-selection is shown in the fig.5.20 by an example of ciphering placeholder. At deployment time, building block pool is searched to find the fitting BBs by matching the ports of BBs and placeholders. Once the suitable BBs are found for a placeholder, they are entered

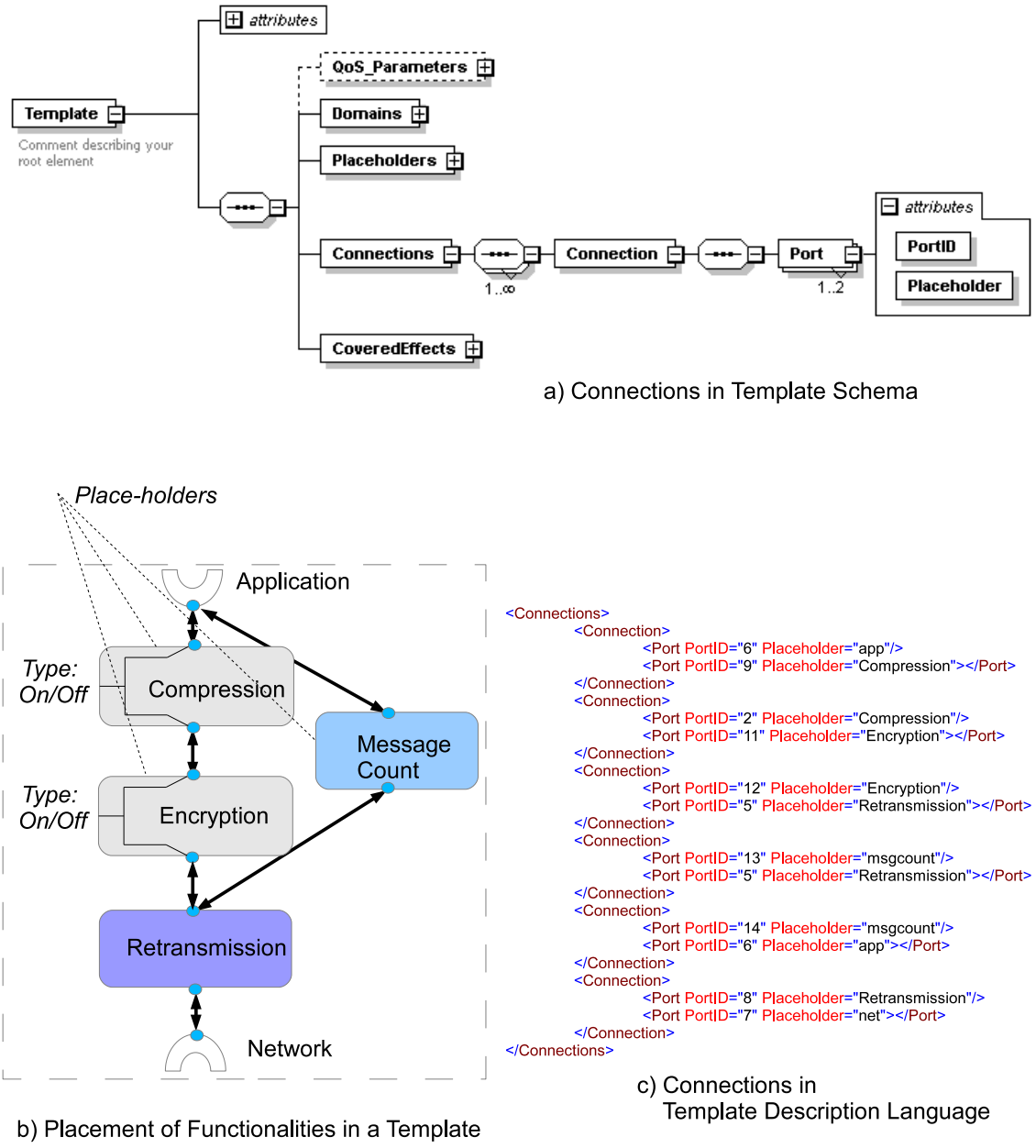


Figure 5.19: Placement in Template

in the pre-selection description of a placeholder as shown in fig. 5.20 (b). This process is done for every placeholder in all the existing templates. This process helps to exclude inapplicable matches before the run-time (i.e., critical time). Hence at the run-time, the selection only takes place within pre-selected BBs as shown in fig. 5.20(c). The selection is further refined by incoming application requirements, policies and network conditions.

So as a conclusion, the split of the composition in different time-phases helps template-

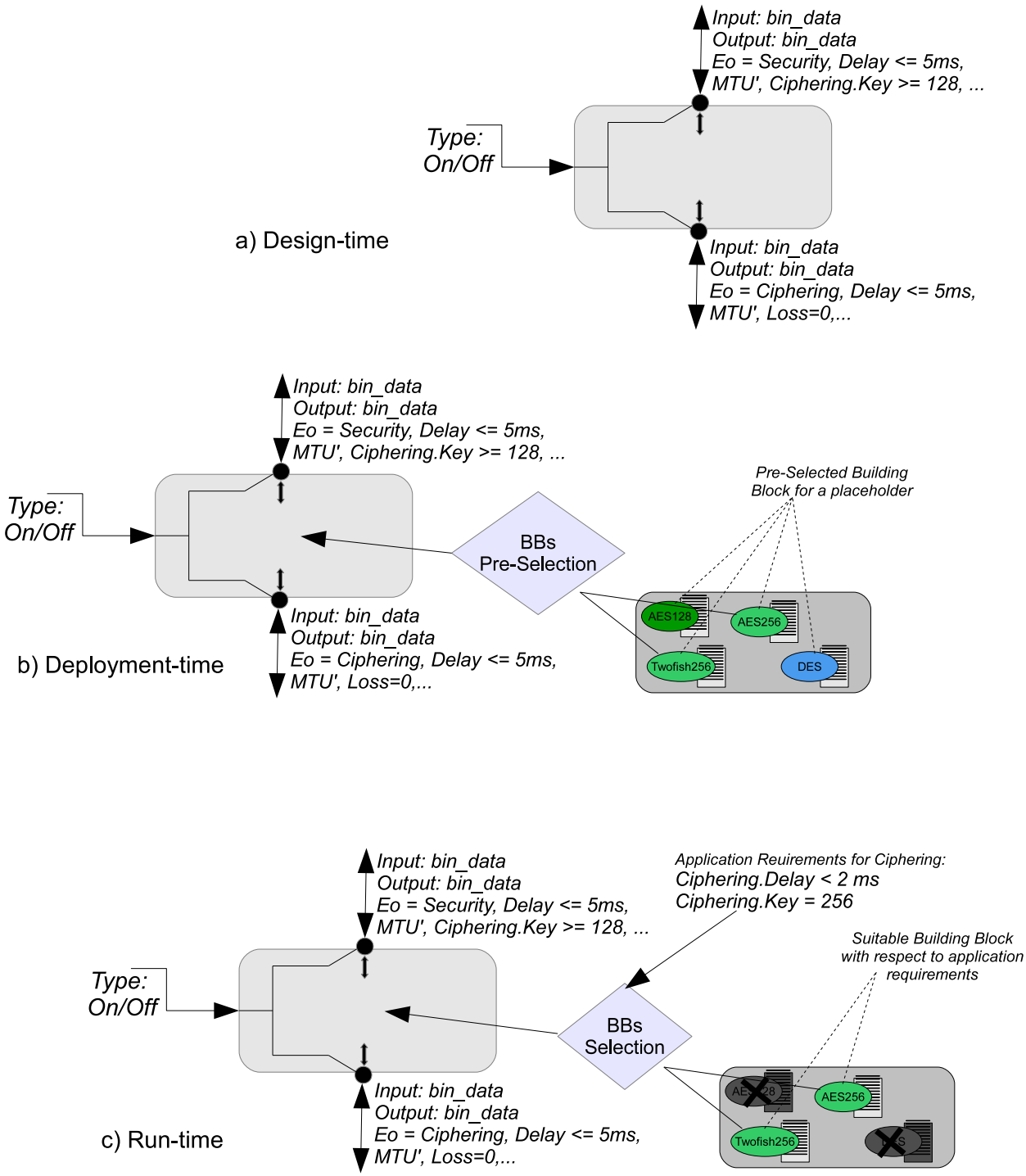


Figure 5.20: Selection of Building Blocks in different Time Phases

based approach to deal with the complex nature of a composition and eventually reduces the required setup time.

Chapter 6

Performance Evaluation of Template Based Composition

The presented approach will not be applicable unless it can be carried out within some time constraints, to ensure the pragmatic nature of it, this chapter focuses on the performance evaluation of the approach. The assessment is based on the time needed *at the*

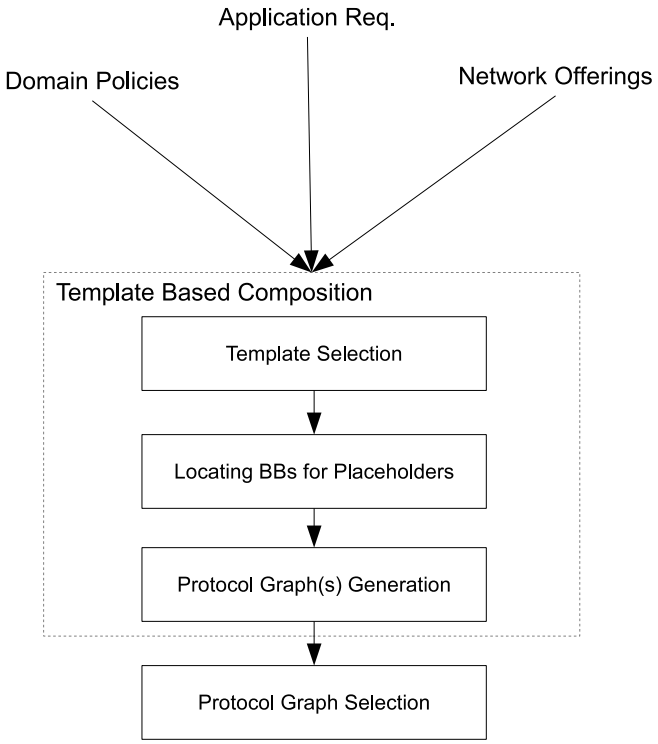


Figure 6.1: Stages of Template Based Composition

run-time to generate the executable protocol graph(s).

The template-based composition is performed in three main stages as shown in figure 6.1, these stages are the template selection, the template filling, and the protocol graph(s) generation. However, this evaluation is based on the actions that are inevitably carried out during the run-time. Hence, processing of building blocks, templates, domain policies and network offerings can be performed before an incoming request. The same is also true for filling the placeholders of templates with suitable BBs. When a new BB is deployed, a background process checks for the matching placeholders and add its reference to the fitting placeholders, thus it does not have an impact on the time required for the generation of protocol graph(s). It leaves remaining two stages "template selection" and "PGs generation". A performance evaluation of those stages is realized in this chapter to calculate the required setup time. In case multiple PGs are generated, one must be selected for execution. The PG selection is also performed at run-time. However, a simple selection method like first-match has no considerable impact on the required time. Besides, it is out of the scope of the composition so that of the presented work.

6.1 Test Environment Specifications

This section lists the hardware, software and technology used for this evaluation.

- **Hardware:** The testing is done using the following hardware.
  - Process: Intel (R) Pentium (R) CPU B970 2.30GHz - Dual Core
  - RAM: 2 GB
- **Software & Technology:** The following is the list of software and technology used during the evaluation.
  - Programming Language: JAVA (JDK 1.7)
  - OS: Windows 7 Ultimate (64 bits) with Service Pack 1

6.2 Selection of Template(s)

The time consumption during the selection process mainly depends on the numbers of incoming requirements and the available templates in a template pool. Moreover, based on the assumption that percentage of suitable templates in a pool will have an impact on how fast the selection process can find the match(es). Different samplings are simulated differ by percentage of available fitting templates for the given requirements.

6.2.1 Single Selection

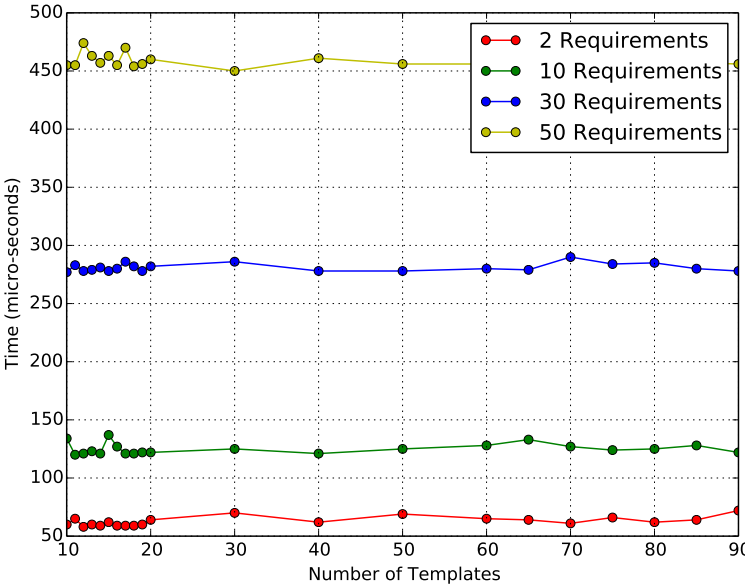
In this simulation only a single template is selected regardless of how many suitable templates are available in a system. It uses the first exact match algorithm. A required time graph is plotted against the number of available templates with a different number of requirements (i.e., 2, 10, 30, 50) and two different percentages of suitable templates available. The amount of BBs and placeholders have no impact on the result as to select a template the requirements are checked only against the covered effects of a template.

The rounded-off number of templates are generated which fulfill the given requirements such as for the 80% suitability if the pool has 22 templates altogether then - 22*80/100 = 17.6 (18) - 18 templates will be suitable ones and rest of them unsuitable. The generated templates are randomly distributed with a random number of matching covered-effects. Hence, there is no fixed number before a suitable template will be found, it can be found anywhere from the first to the last template. The randomly covered effects also have an impact, if an unsuitable template covers 18 requirements out of 20 then it can be that first checked requirement is unfitting or the second last, both of these cases will consume different time.

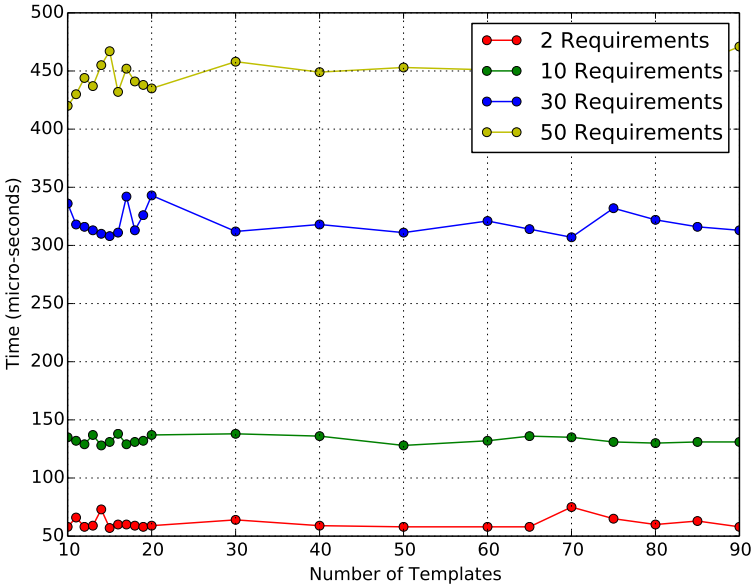
It is apparent from both of these graphs 6.2a 6.2b that as the number of requirements increase so the taken time. The main reason behind is the algorithm complexity, as shown in Listing 6.2.1. Two loops are being used which makes it n^2, however, the inner loop is probabilistic and gives a constant number, which makes the big-o-notation to simple "n" which represents a number of requirements. Hence, the number of templates and its distribution has the least impact on the performance of the selection process.

```
for (Requirement req : requirements) {
        for (Template temp: templates)
        {
        ...
        }
}
```

The time is measured in the microseconds, the average maximum required time is between 400 and 500 microseconds with some deviation because of a random sampling of templates and a number of suitably covered effects associated. It can also be concluded from the plotted graphs that major performance effecting parameter is number of requirements during the selection of a single template.



(a) Single Selection With 10% Matching Templates



(b) Single Selection With 50% Matching Templates

6.2.2 Multiple Selection

In this simulation all the possible matching templates are selected with varying number of requirements and number of available templates. For a selection of all possible matches, the selection process must go through all the available templates and select whichever are fitting to the given requirements, which consequently increases the required time.

This simulation uses four different percentages of suitable templates. Some requirements have the major impact on the required time same as in the previous case, but unlike single selection, the number of templates has significance in overall performance. All the graphs presented in the figure 6.3 have linear growth with respect to number of templates. It is also to be noted that suitability also plays a role in this case as the required-time increases with the number of suitable templates. However, the main factor is to go through all the templates before finding all the matching ones. In this case, the process requires checking all the available templates before finding all matching ones unlike the previous case where it terminates as soon a single suitable template is found. Selection of suitable one requires longer than rejection, as during the selection, the process must check the covered effects of a template. But in another case, a template is rejected as soon as the single requirement is not fulfilled.

The time taken for the extreme case of 50 requirements and 90 available template is between 20 and 23 Milliseconds. With the same number of requirements and change in

the number of templates, the ascending is almost stable as shown in graph 6.3a with 10 requirements. The addition of every template increases the time on average by 40 to 50 microseconds and in case of 30 requirements by 100 to 200 microseconds.

6.3 Protocol Graph(s) Generation

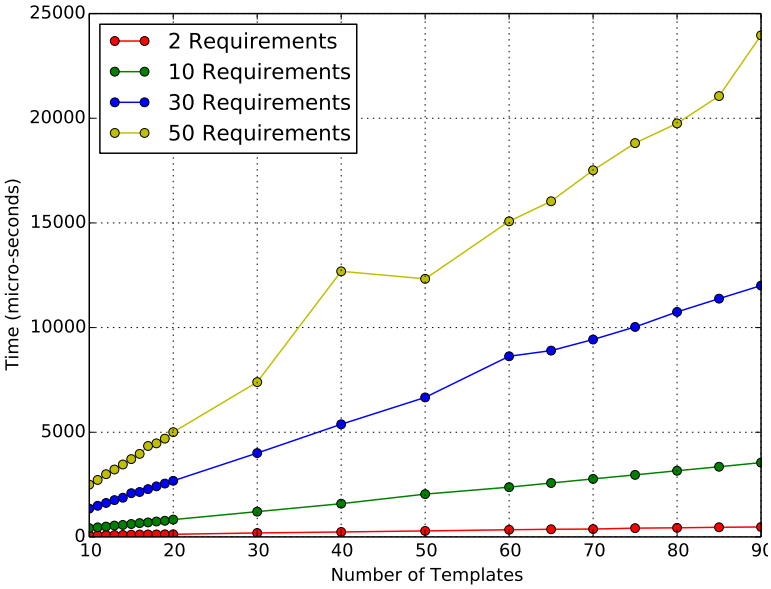
The next run-time activity is to replace placeholders of a template with actual implementation (BBs) to generate an executable protocol graph. The process involves to connects BBs' ports to placeholders' ports and place them in order. It also aggregates QoS parameters of all the BBs in a protocol graph for further optimal selection in case more than one PGs are generated.

The required time mainly depend on the number of placeholders and the available suitable BBs for each placeholder, which accounts for number of PGs. The simulation is divided into two sub-cases, one where only single PG is generated and second where all possible PGs are created. Multiple PGs are generated to select the most optimal PG on the given QoS requirements. However, in many cases, even having a single executable PG will be sufficient which is measured in the first case. The second case generates every possible executable PG so that later, a best possible PG will be selected for service provision.

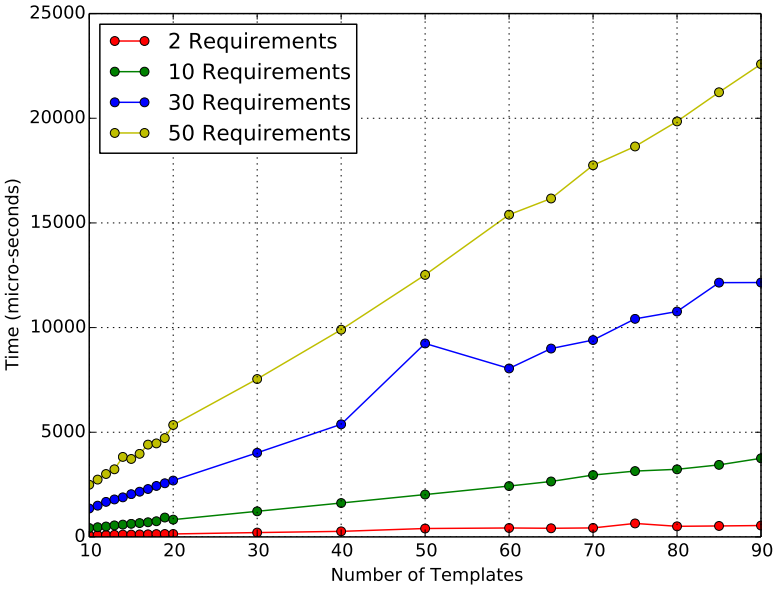
6.3.1 Single Protocol Graph

In this simulation for the constant increase of resultant PGs, it is assumed that every placeholder in a template has the same amount of suitable BBs, which is unlikely in everyday scenarios as contemporary number of security implementations is higher than flow control implementations. In this case, the PG generation process breaks after creating an executable PG successfully.

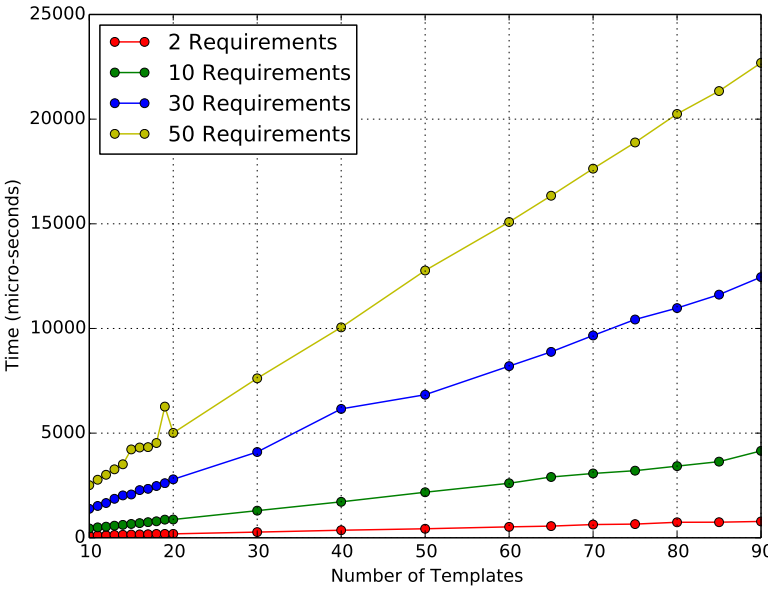
It is apparent from the graph 6.4 that number of placeholders plays a vital role in the increment of required time. The effect on the time is minimal on accretion in some BBs, it is more evident in case 10 BBs but yet negligible. The required time is between 200 and 300 microseconds for 6 placeholders, hence the difference between 1 suitable BB and 10 BBs for 6 placeholders is $\sim$100 microseconds.



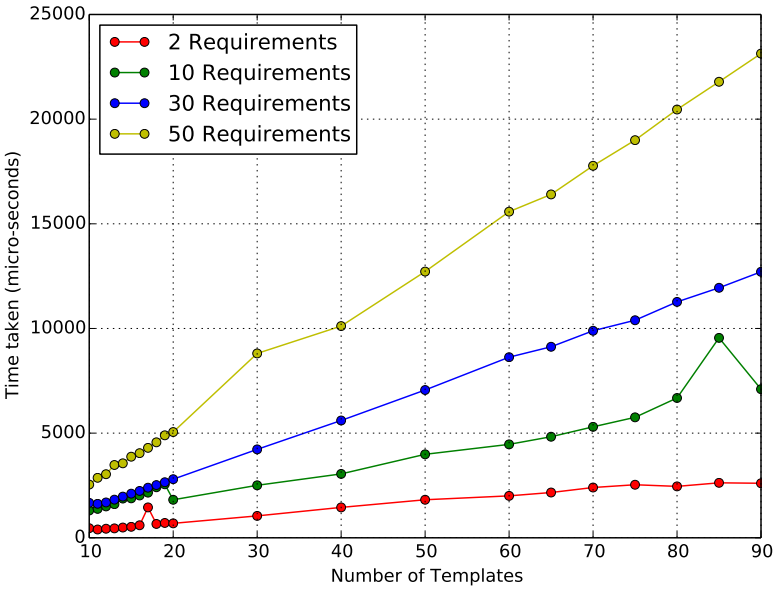
(a) All Possible Matches With 10% Matching Templates



(b) All Possible Matches With 20% Matching Templates



(c) All Possible Matches With 50% Matching Templates



(d) All Possible Matches With 80% Matching Templates

Figure 6.3: Multi-Template Selection

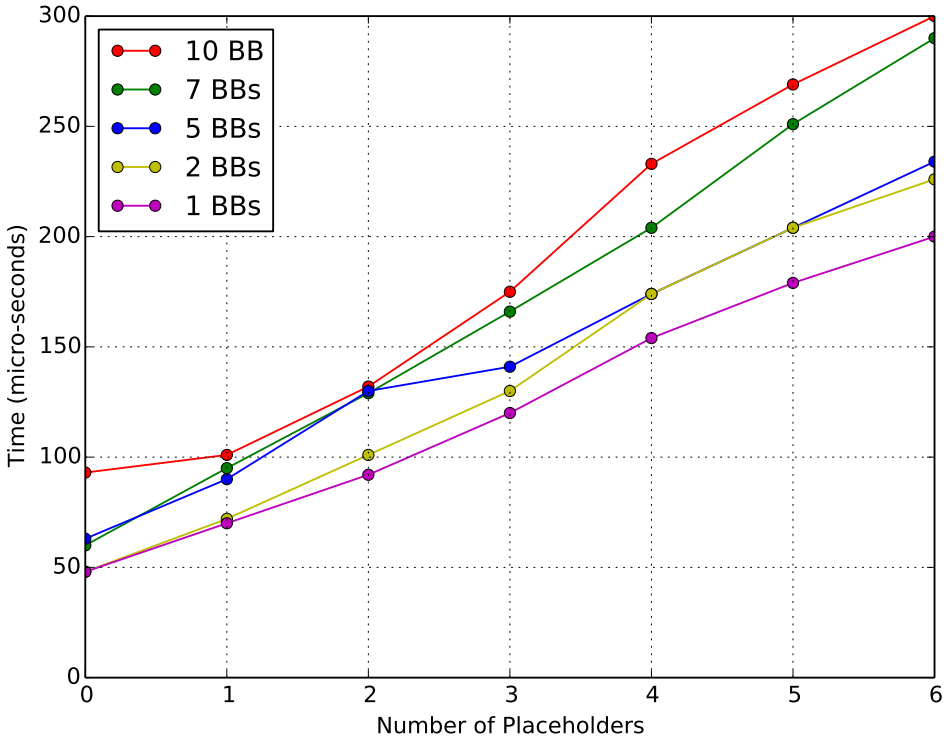


Figure 6.4: Graph: Time Required to Produce an Executable PG

6.3.2 All Possible Protocol Graphs

In this case, like in the previous scenario6.3.1, every placeholder in a template holds the same number of suitable BBs, but here, all possible protocol graphs are generated.

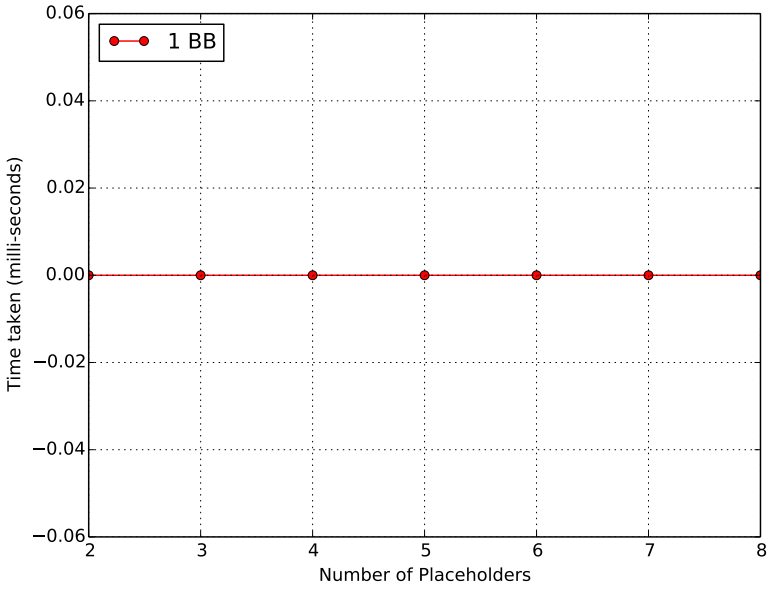
The possible number of PGs in a template is calculated by following formula:

$PossiblePGs = \prod_{i=1}^{n} PB_i$

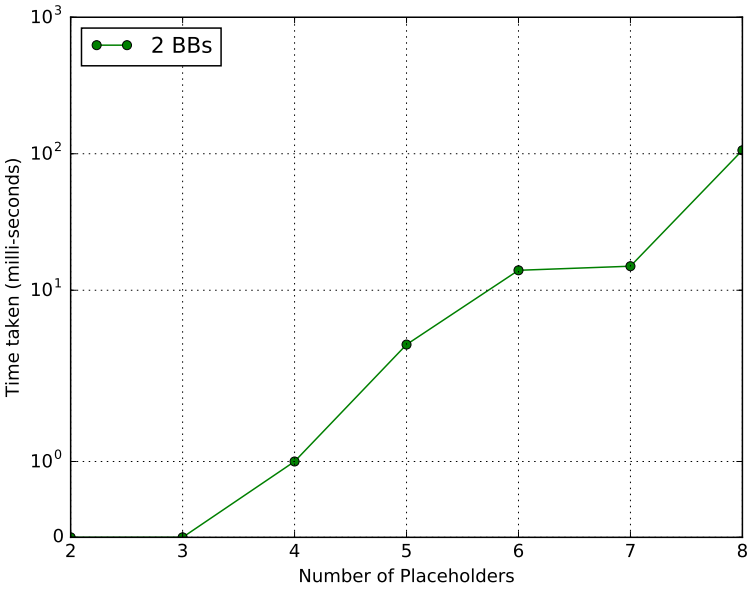
Where PB_i is number of BBs in the placeholder-i.

However, as in this simulation, every placeholder contains the exact same amount of suitable BBs it can simply be calculated by $PossiblePGs = (NumberofBBs)^{(NumberofPlaceholders)}$.

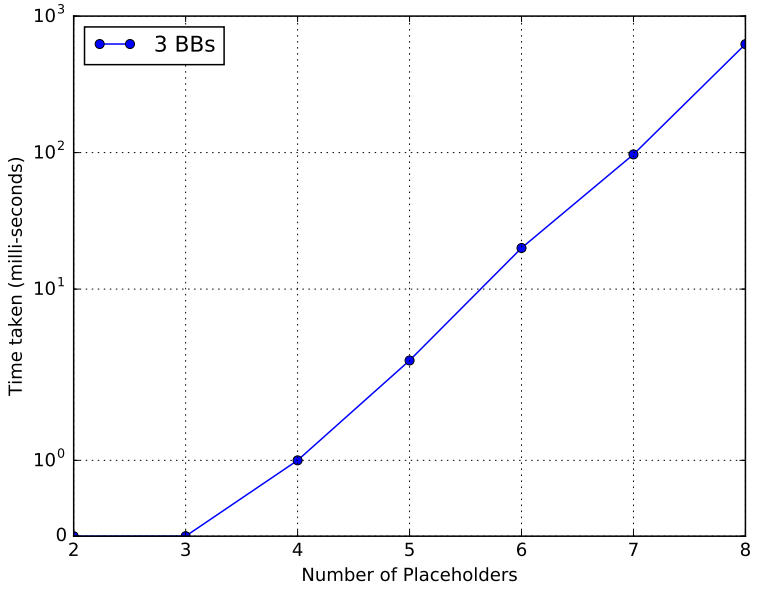
The four graphs 6.5 are drawn for a different number of building blocks as the value varies significantly (0-6000 Milliseconds) to visualize all the readings in one graph. It is also important to note that unlike other graphs, the time is measured in Milliseconds. In this scenario, a number of BBs and placeholders both play a vital role in the consumed time, as it is more about how many PGs are generated as a result. In the worst-case scenario with 4 BBs and 8 placeholders, 65536 PGs are generated which takes about 6



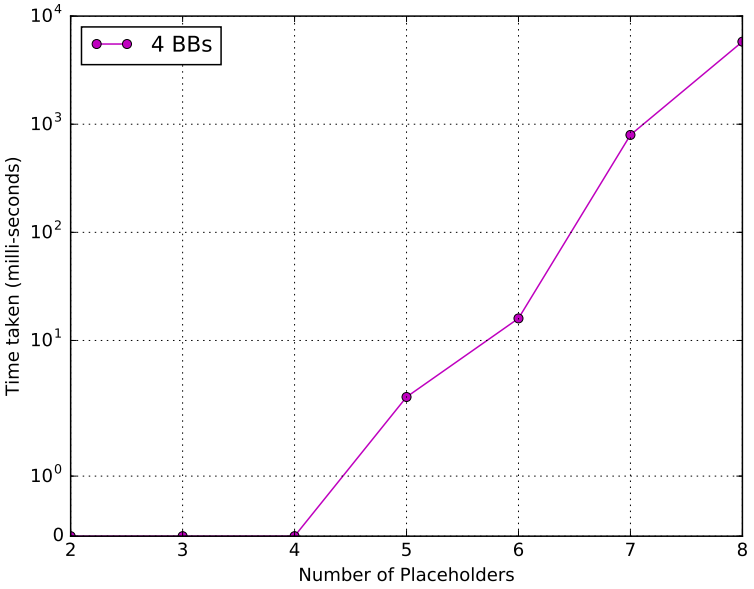
(a) 1 Suitable BB



(b) 2 Suitable BBs



(c) 3 Suitable BBs



(d) 4 Suitable BBs

Figure 6.5: All Possible PGS With Different Suitable BBs In Each Placeholder

seconds (about a second for every 1100 PGs).

6.4 Performance Conclusion

The presented performance evaluation shows that the use of the approach will highly depend on the number of requirements, number of BBs and time criticality of a domain. However, it requires less than a millisecond to generate a single executable PG as demonstrated in this evaluation. Every generated PG is executable though not optimal (i.e., as not selected based on QoS requirements) on the given requirements. An executable protocol graph with 10 requirements and 6 placeholders can be generated within 200 microseconds. The worst case scenario (i.e., 65536 PGs), presented in 6.3.2, can take up to few seconds, which will not be suitable for domains where required-setup time is less than a second. However, as a research prototype, the current implementation is not exclusively focused on performance and uses verbose but extensible meta-language like XML. It can be enhanced with performance oriented implementation besides use of more concise data representations like JSON or other binary forms. Besides, the incoming requirements and the generated PGs are not cached, which can also be a performance-improving feature to avoid the redundant processing. The selection of most optimal PG out of generated PGs is not part of the presented work.

Chapter 7

Conclusion

The presented approach addresses the challenge of trade-off between time-complexity, regarding setup time, and desired flexibility in a functional composition. As a result, a template based composition is presented and successfully tested [GSS+12] within the framework of G-Lab project[1]. The approach splits the composition process in different time-phases to reduce the setup-time and yet to provide the desired flexibility for future network architectures.

The approach also assures that every generated protocol graph is executable by having a pre-defined connections (i.e., template) among the functionalities and later replacing it with an actual implementation. The selection of most suitable one is required if multiple PGs are generated this work does not cover this part. However, a simple multiple criteria decision analysis approach like Analytical Hierarchy Process can be used to select a better suited PG out of the generated pool. How much time it requires to choose a better-suited graph has a direct impact on the required setup time. But to remain independent of any additional selection process, a first-match algorithm has been used for evaluation that requires no time for the selection and it still provides the desired service though not best suited one.

The current implementation requires experts to develop the templates but an automatic process can also create it. The approach has its limitations regarding the required time to generate multiple PGs with increment in a number of functionalities and possible suitable implementations (i.e., BBs) as evaluated in 6.3.2. Hence, it will depend on the domain specifications and constraints that if the performance of the approach is suitable for that particular domain.

This work presents a general composition approach with the focus on domain of net-

[1] http://www.german-lab.de/

work architectures. The presented approach can be used independently of a network architecture in other domains like overlay networks, and Web-service composition. However, it may require modifications in descriptions (e.g., service, policies, requirements) and domain related ontologies. The well-defined ordering of abstract functionalities (i.e., placeholders) makes the approach adaptable to needs of other domains. The following section describes the future work, which can be undertaken to cover another aspect of the composition process.

Future Work

This section describes the work, which can be carried out in future to cover the aspects of the future Internet architectures that have not been yet covered in SONATE or the presented work. The following text first describes adaptation in the template, which is to change a BB to adapt to network changes during the running communication. Next, the text describes the heterogeneity. Heterogeneity is the diversity of the services within inter- and intra-architecture.

Adaption within Template

The functional composition is a process, which is performed before the actual communication takes place so that it is assumed that application/user requirements do not change during communication. However, what happens if network conditions or some other policies change during a communication? The reaction to that change is referred as adaption in this text. It is a valuable feature in mobile communication wherein the same session a mobile user switches from one network to another and experiences the different network conditions, which are needed to be adapted.

The adaption of the functional composition is not to create a new protocol graph but how some functionalities can be turned-off or turned-on without creating any malfunction during a communication. An alternative would be to create a new connection based on new requirements. However, it requires to save the status information like packet numbering and connection duration or to re-initiate the connection. And these additional processes will increase the overall delay and management complexity. Hence, it is chosen to limit the adaption by toggling the functionalities during the run-time. The abstraction of placeholder comes as handy to deal with this feature.

Toggling a placeholder performs the adaptation in the Template Based FC. A place-

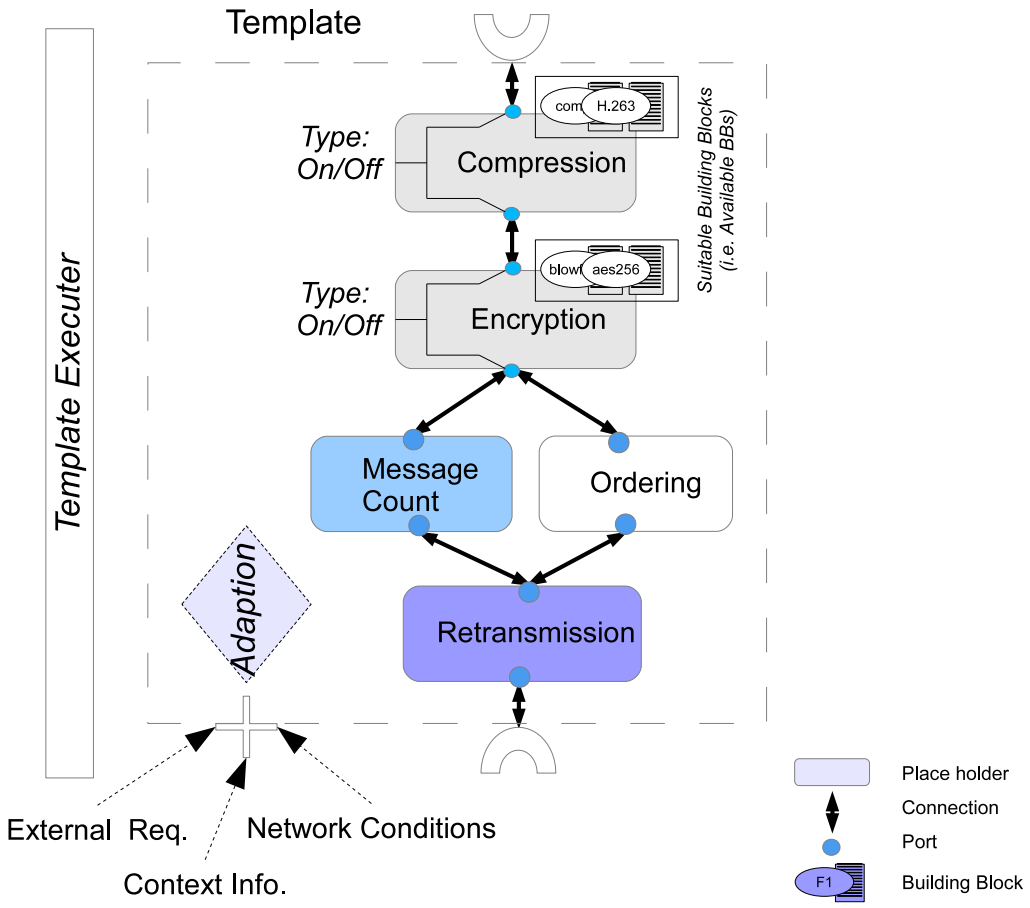


Figure 7.1: Adaption within Template

holder is not randomly toggled, rather it is carefully specified by a designer of a template that which placeholders can be toggled. It is necessary because turning on/off an arbitrary functionality can be responsible for a malfunction or a total failure of a generated protocol graph. Only compression and encryption functionalities can be toggled in the given template as shown in figure 7.1.

To perform a run-time adaption within a template has many necessities some of those are mentioned below.

- A workflow engine must be able to execute a template rather than a protocol graph (i.e., based on fixed BBs). If a protocol graph is executed instead of a template then given abstraction of a placeholder cannot be used.

- A template needs an interface for receiving external requirements, network constraints and varying contexts at the run-time as shown in the fig. 7.1.

- It is also needed to have an adaption mechanism which goes through the inputs and adapt a template accordingly.
- A place-holder must provide an interface to turn on/off a specific functionality of a place-holder.
- It is also necessary to signal the change to communicating partner, so that it is known to both parties, which functionality is on or off and which BB for the specific functionality is selected.

How adaption works: After the selection of templates, only the placeholders will be turned on which have been requested by the requirements given that a placeholder can be toggled. Once a template has been selected, suitable building blocks will be filled for all the placeholders in that template regardless of their status (i.e., on or off). Later, this selected template will be given to a workflow engine to execute.

The run-time adaption occurs once a workflow engine is executing a template. Let's assume that compression was part of the selected template but it was not turned on as it was not needed on the given network conditions (i.e., sufficient bandwidth so no compression) or domain policies (e.g., If the domain is data transmission then compression is not included). Now let's assume that the given bandwidth changes drastically which add a new requirement on the template to enable the compression and to use best possible BB. At this point, the adaption mechanism receives the new requirements via given interface to enable the compression functionality. Now it is the task of the adaption mechanism to search for the placeholder(s) that fulfill those requirements and to enable it accordingly. However, enabling a functionality is not the whole story, it is also needed to select out of possible suitable BBs. The simplest way to do this to use the BB that is specified as standard and already connected. However, it is not enough just to select the most suitable BB but it is also necessary to signal the occurred changes to the communicating partner so that same changes can take place at the other side.

Like during composition, we may also face the **heterogeneity** (i.e., see section 2.3.7) problem, it might be that the other end does not support the selected BB or this specific BB is not available on the system. The difference here is that connection in use might be disconnected. Specifying a most used BB as standard BB for this functionality can solve this problem. For instance aes256 can be used as a standard BB for encryption functionality. By this method, we can select a standard BB and can also suggest other more preferred BBs in a prioritize order to the other end. Depending on the availability, the other end can decide for the one of the preferred BBs or in a worst case it will fall-back

to the standard BB. However, the practical implementation of any of this is not part of the current work.

The current work supports the idea of adaption by providing the toggling property to placeholders in the description language. The implementation of this adaption feature can be done in future, and the mobile-domain can be used as a testing scenario.

Heterogeneity of Services

The flexible composition of protocol graphs will create even further heterogeneity. The heterogeneity is described briefly in section 2.3.7, however not being the focus of the presented work, it has not been researched in detail. This published paper "Mediation between Service and Network Composition [Sea10b]" describes two kind of negotiation schemes so-called implicit and explicit. Wherein implicit negotiation, one end uses the cache to decide for the suitable protocol graph for the other end. And in explicit negotiation, one end communicates the used protocol graph before connection establishment. To implement these negotiation schemes besides finding others will be beneficial for future network architectures.

Appendices

Appendix A

Examples & Implementation

A.1 Requirements

Listing A.1.1: Requirements with Retransmission

```
<?xml version="1.0" encoding="UTF-8"?>
<Requirements xmlns:xsi="Req_Desc.xsd">
        <Domain>ImageTransmission</Domain>
        <Requirement>
                <Effect>retransmission</Effect>
                <Operator>=</Operator>
                <Attribute>true</Attribute>
        </Requirement>
</Requirements>
```

Listing A.1.2: Requirements with Security and Reliability

```
<?xml version="1.0" encoding="UTF-8"?>
<Requirements xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:
    noNamespaceSchemaLocation="Req_Desc.xsd">
        <Domain>ImageTransmission</Domain>
        <Requirement>
                <Effect>ciphering</Effect>
                <Operator>=</Operator>
                <Attribute>true</Attribute>
        </Requirement>
        <Requirement>
                <Effect>retransmission</Effect>
                <Operator>=</Operator>
                <Attribute>true</Attribute>
        </Requirement>
</Requirements>
```

Listing A.1.3: Example of Requirements

```
<?xml version="1.0" encoding="UTF-8"?>
<Requirements xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:
    noNamespaceSchemaLocation="Req_Desc.xsd">
        <Domain>ImageTransmission</Domain>
        <Requirement>
                <Effect>ciphering</Effect>
                <Operator>=</Operator>
                <Attribute>true</Attribute>
```

```
        </Requirement>
        <Requirement>
                <Effect>retransmission</Effect>
                <Operator>=</Operator>
                <Attribute>true</Attribute>
        </Requirement>
        <Requirement>
                <Effect>MessageCount</Effect>
                <Operator>=</Operator>
                <Attribute>true</Attribute>
        </Requirement>
        <Requirement>
                <Effect>DataReduction</Effect>
                <Operator>=</Operator>
                <Attribute>true</Attribute>
        </Requirement>
</Requirements>
```

Listing A.1.4: Example of Quality of Transmission Domain Requirement

```
<?xml version="1.0" encoding="UTF-8"?>
<Requirements xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:
    noNamespaceSchemaLocation="Req_Desc.xsd">
        <Domain>QualityTransmission</Domain>
        <Requirement>
                <Effect>retransmission</Effect>
                <Operator>=</Operator>
                <Attribute>true</Attribute>
        </Requirement>
</Requirements>
```

Listing A.1.5: Data Transmission Domain Requirements

```
<?xml version="1.0" encoding="UTF-8"?>
<Requirements xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:
    noNamespaceSchemaLocation="Req_Desc.xsd">
        <Domain>SecureReilableDataTransmission</Domain>
        <Requirement>
                <Effect>retransmission</Effect>
                <Operator>=</Operator>
                <Attribute>true</Attribute>
        </Requirement>
        <Requirement>
                <Effect>ciphering</Effect>
                <Operator>=</Operator>
                <Attribute>true</Attribute>
        </Requirement>
</Requirements>
```

A.2 Domain Policies

Listing A.2.1: Example of Domain Policies

```
<?xml version="1.0" encoding="UTF-8"?>
<DomainsPolicies xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:
    noNamespaceSchemaLocation="DomainPolicies.xsd">
        <Domain Name="DataTransmission">
        <Condition>
                <IF Effect="bandwidth" Operator="=&lt;" Attribute="2" Unit="MB"
                    ></IF>
                <Then Effect="DataReduction" Operator="=" Attribute="true"></
                    Then>
        </Condition>
        </Domain>
        <Domain Name="ImageTransmission">
```

```
        <Condition>
                <IF Effect="bandwidth" Operator="=&lt;" Attribute="1" Unit="MB"
                    ></IF>
                <Then Effect="DataReduction" Operator="=" Attribute="true"></
                    Then>
        </Condition>
        </Domain>
        <Domain Name="QualityTransmission">
        <Requirement>
                <Effect>retransmission</Effect>
                <Operator>=</Operator>
                <Attribute>true</Attribute>
        </Requirement>
        </Domain>
        <Domain Name="SecureReilableDataTransmission">
        <Requirement>
                <Effect>retransmission</Effect>
                <Operator>=</Operator>
                <Attribute>true</Attribute>
        </Requirement>
        <Requirement>
                <Effect>Ordering</Effect>
                <Operator>=</Operator>
                <Attribute>true</Attribute>
        </Requirement>
        <Requirement>
                <Effect>Congestion</Effect>
                <Operator>=</Operator>
                <Attribute>true</Attribute>
        </Requirement>
        <Requirement>
                <Effect>Flow</Effect>
                <Operator>=</Operator>
                <Attribute>true</Attribute>
        </Requirement>
        <Requirement>
                <Effect>Connection</Effect>
                <Operator>=</Operator>
                <Attribute>true</Attribute>
        </Requirement>
        <Requirement>
                <Effect>Error Detection</Effect>
                <Operator>=</Operator>
                <Attribute>true</Attribute>
        </Requirement>
        <Requirement>
                <Effect>Multiplexing</Effect>
                <Operator>=</Operator>
                <Attribute>true</Attribute>
        </Requirement>
        </Domain>
</DomainsPolicies>
```

A.3 Templates

Listing A.3.1: Template for Secure Reliable Data Transmission

```
<?xml version="1.0" encoding="UTF-8"?>
<Template xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:
    noNamespaceSchemaLocation="file/Template_Description.xsd">
  <Domains>
    <Domain>SecureReilableDataTransmission</Domain>
  </Domains>
  <Placeholders>
    <PlaceHolder Name="app" ID="3">
      <ToggleEnable isToggle="false" />
      <Port PortID="6">
        <OfferedEffect Effect="application" Operator="=" Attribute="true" />
      </Port>
```

```
</PlaceHolder>
<PlaceHolder Name="MuxDeMux" ID="1">
  <ToggleEnable isToggle="true" />
  <Port PortID="11">
    <OfferedEffect Effect="mux" Operator="=" Attribute="true" />
  </Port>
  <Port PortID="12">
    <OfferedEffect Effect="demux" Operator="=" Attribute="true" />
  </Port>
  <Port PortID="13">
    <OfferedEffect Effect="Mangement" Operator="=" Attribute="true" />
  </Port>
</PlaceHolder>
<PlaceHolder Name="Connection" ID="11">
  <ToggleEnable isToggle="false" />
  <Port PortID="111">
    <OfferedEffect Effect="req" Operator="=" Attribute="true" />
  </Port>
  <Port PortID="112">
    <OfferedEffect Effect="reply" Operator="=" Attribute="true" />
  </Port>
</PlaceHolder>
<PlaceHolder Name="Compression" ID="2">
  <ToggleEnable isToggle="true" />
  <Port PortID="9">
    <OfferedEffect Effect="Data" Operator="=" Attribute="true" />
  </Port>
  <Port PortID="2">
    <OfferedEffect Effect="DataReduction" Operator="=" Attribute="true" />
  </Port>
</PlaceHolder>
<PlaceHolder Name="Encryption" ID="5">
  <ToggleEnable isToggle="true" />
  <Port PortID="51">
    <OfferedEffect Effect="Data" Operator="=" Attribute="true" />
  </Port>
  <Port PortID="52">
    <OfferedEffect Effect="ciphering" Operator="=" Attribute="true" />
  </Port>
</PlaceHolder>
<PlaceHolder Name="CRC" ID="12">
  <ToggleEnable isToggle="true" />
  <Port PortID="121">
    <OfferedEffect Effect="Data" Operator="=" Attribute="true" />
  </Port>
  <Port PortID="122">
    <OfferedEffect Effect="Integrity" Operator="=" Attribute="true" />
  </Port>
</PlaceHolder>
<PlaceHolder Name="Packetization" ID="7">
  <ToggleEnable isToggle="false" />
  <Port PortID="71">
    <OfferedEffect Effect="Data" Operator="=" Attribute="true" />
  </Port>
  <Port PortID="72">
    <OfferedEffect Effect="Packets" Operator="=" Attribute="0" />
  </Port>
  <Port PortID="73">
    <OfferedEffect Effect="Mangement" Operator="=" Attribute="0" />
  </Port>
</PlaceHolder>
<PlaceHolder Name="Retransmission" ID="4">
  <ToggleEnable isToggle="false" />
  <Port PortID="5">
    <OfferedEffect Effect="Data" Operator="=" Attribute="true" />
  </Port>
  <Port PortID="8">
    <OfferedEffect Effect="LossRatio" Operator="=" Attribute="0" />
  </Port>
</PlaceHolder>
<PlaceHolder Name="Control" ID="10">
  <ToggleEnable isToggle="false" />
  <Port PortID="101">
    <OfferedEffect Effect="Data" Operator="=" Attribute="true" />
```

```
      </Port>
      <Port PortID="102">
        <OfferedEffect Effect="Controlling" Operator="=" Attribute="0" />
      </Port>
    </PlaceHolder>
    <PlaceHolder Name="net" ID="6">
      <ToggleEnable isToggle="false" />
      <Port PortID="7">
        <OfferedEffect Effect="net" Operator="=" Attribute="true" />
      </Port>
    </PlaceHolder>
  </Placeholders>
  <Connections>
    <Connection>
      <Port PortID="6" Placeholder="app" />
      <Port PortID="11" Placeholder="MuxDeMux" />
    </Connection>
    <Connection>
      <Port PortID="12" Placeholder="MuxDeMux" />
      <Port PortID="9" Placeholder="Compression" />
    </Connection>
    <Connection>
      <Port PortID="13" Placeholder="MuxDeMux" />
      <Port PortID="111" Placeholder="Connection" />
    </Connection>
    <Connection>
      <Port PortID="2" Placeholder="Compression" />
      <Port PortID="51" Placeholder="Encryption" />
    </Connection>
    <Connection>
      <Port PortID="52" Placeholder="Encryption" />
      <Port PortID="71" Placeholder="Packetization" />
    </Connection>
    <Connection>
      <Port PortID="111" Placeholder="Connection" />
      <Port PortID="73" Placeholder="Packetization" />
    </Connection>
    <Connection>
      <Port PortID="72" Placeholder="Packetization" />
      <Port PortID="5" Placeholder="Retransmission" />
    </Connection>
    <Connection>
      <Port PortID="8" Placeholder="Retransmission" />
      <Port PortID="121" Placeholder="CRC" />
    </Connection>
   <Connection>
      <Port PortID="122" Placeholder="CRC" />
      <Port PortID="101" Placeholder="Control" />
    </Connection>
    <Connection>
      <Port PortID="102" Placeholder="Control" />
      <Port PortID="7" Placeholder="net" />
    </Connection>
  </Connections>
  <CoveredEffects>
     <CoveredEffect Effect="Congestion" Operator="=" Attribute="␣true" isToggle
        ="true" />
    <CoveredEffect Effect="Flow" Operator="=" Attribute="␣true" isToggle="false
        " />
    <CoveredEffect Effect="Connection" Operator="=" Attribute="␣true" isToggle=
        "false" />
    <CoveredEffect Effect="Error␣Detection" Operator="=" Attribute="␣true"
        isToggle="true" />
    <CoveredEffect Effect="Multiplexing" Operator="=" Attribute="␣true"
        isToggle="false" />
    <CoveredEffect Effect="retransmission" Operator="=" Attribute="␣true"
        isToggle="false" />
    <CoveredEffect Effect="ciphering" Operator="=" Attribute="␣true" isToggle="
        true" />
    <CoveredEffect Effect="DataReduction" Operator="=" Attribute="␣true"
        isToggle="true" />
  </CoveredEffects>
</Template>
```

A.4 Protocol Graphs

Listing A.4.1: Protocol Graph for Secure Reliable Data Transmission

```
<?xml version="1.0" encoding="UTF-8"?>
<Workflow>
   <Optional>
      <Offering>
         <Effect>delay</Effect>
         <Operator>=</Operator>
         <Attribute Unit="">85.0</Attribute>
      </Offering>
      <Offering>
         <Effect>imagequality</Effect>
         <Operator>=</Operator>
         <Attribute Unit="">75.0</Attribute>
      </Offering>
      <Offering>
         <Effect>bandwidth</Effect>
         <Operator>=</Operator>
         <Attribute Unit="">0.0</Attribute>
      </Offering>
      <Offering>
         <Effect>lossratio</Effect>
         <Operator>=</Operator>
         <Attribute Unit="">0.0</Attribute>
      </Offering>
   </Optional>
   <buildingblocks>
      <buildingblock id="app" uuid="app" special="app">
         <Port PortID="data">
            <OfferedEffect Effect="application" Operator="=" Attribute="true"
                />
         </Port>
      </buildingblock>
      <buildingblock id="Mux" uuid="Mux">
         <Port PortID="down">
            <OfferedEffect Effect="demux" Operator="=" Attribute="true" />
         </Port>
         <Port PortID="extra">
            <OfferedEffect Effect="Mangement" Operator="=" Attribute="true" />
         </Port>
         <Port PortID="up">
            <OfferedEffect Effect="mux" Operator="=" Attribute="true" />
         </Port>
      </buildingblock>
      <buildingblock id="Management" uuid="Management">
         <Port PortID="down">
            <OfferedEffect Effect="req" Operator="=" Attribute="true" />
         </Port>
         <Port PortID="up">
            <OfferedEffect Effect="reply" Operator="=" Attribute="true" />
         </Port>
      </buildingblock>
      <buildingblock id="ImageComp75" uuid="ImageComp75">
         <Port PortID="down">
            <OfferedEffect Effect="DataReduction" Operator="=" Attribute="true"
                />
         </Port>
         <Port PortID="up">
            <OfferedEffect Effect="Data" Operator="=" Attribute="true" />
         </Port>
      </buildingblock>
      <buildingblock id="AES256" uuid="AES256">
         <Port PortID="down">
            <OfferedEffect Effect="ciphering" Operator="=" Attribute="true" />
         </Port>
         <Port PortID="up">
            <OfferedEffect Effect="Data" Operator="=" Attribute="true" />
         </Port>
      </buildingblock>
```

```
    <buildingblock id="CRC" uuid="CRC">
       <Port PortID="down">
          <OfferedEffect Effect="Integrity" Operator="=" Attribute="true" />
       </Port>
       <Port PortID="up">
          <OfferedEffect Effect="Data" Operator="=" Attribute="true" />
       </Port>
    </buildingblock>
    <buildingblock id="Fragementation" uuid="Fragementation">
       <Port PortID="down">
          <OfferedEffect Effect="Data" Operator="=" Attribute="true" />
       </Port>
       <Port PortID="extra">
          <OfferedEffect Effect="Mangement" Operator="=" Attribute="true" />
       </Port>
       <Port PortID="up">
          <OfferedEffect Effect="Packets" Operator="=" Attribute="true" />
       </Port>
    </buildingblock>
    <buildingblock id="Retransmission" uuid="Retransmission">
       <Port PortID="up">
          <OfferedEffect Effect="Data" Operator="=" Attribute="true" />
       </Port>
       <Port PortID="down">
          <OfferedEffect Effect="LossRatio" Operator="=" Attribute="0" />
       </Port>
    </buildingblock>
    <buildingblock id="FlowCongControl" uuid="FlowCongControl">
       <Port PortID="down">
          <OfferedEffect Effect="Data" Operator="=" Attribute="true" />
       </Port>
       <Port PortID="up">
          <OfferedEffect Effect="Controlling" Operator="=" Attribute="true"
              />
       </Port>
    </buildingblock>
    <buildingblock id="net" uuid="net" special="net">
       <Port PortID="data">
          <OfferedEffect Effect="net" Operator="=" Attribute="true" />
       </Port>
    </buildingblock>
 </buildingblocks>
 <connections>
    <connection>
       <port blockname="app.xml" blockid="app" name="data" id="data" />
       <port blockname="Mux.xml" blockid="Mux" name="up" id="up" />
    </connection>
    <connection>
       <port blockname="Mux.xml" blockid="Mux" name="down" id="down" />
       <port blockname="ImageComp75.xml" blockid="ImageComp75" name="up" id="
          up" />
    </connection>
    <connection>
       <port blockname="Mux.xml" blockid="Mux" name="extra" id="extra" />
       <port blockname="Management.xml" blockid="Management" name="down" id="
          down" />
    </connection>
    <connection>
       <port blockname="ImageComp75.xml" blockid="ImageComp75" name="down" id
          ="down" />
       <port blockname="AES256.xml" blockid="AES256" name="up" id="up" />
    </connection>
    <connection>
       <port blockname="AES256.xml" blockid="AES256" name="down" id="down" />
       <port blockname="Fragementation.xml" blockid="Fragementation" name="
          down" id="down" />
    </connection>
    <connection>
       <port blockname="Management.xml" blockid="Management" name="down" id="
          down" />
       <port blockname="Fragementation.xml" blockid="Fragementation" name="
          extra" id="extra" />
```

```
      </connection>
      <connection>
         <port blockname="Fragementation.xml" blockid="Fragementation" name="up
             " id="up" />
         <port blockname="Retransmission.xml" blockid="Retransmission" name="up
             " id="up" />
      </connection>
      <connection>
         <port blockname="Retransmission.xml" blockid="Retransmission" name="
             down" id="down" />
         <port blockname="CRC.xml" blockid="CRC" name="up" id="up" />
      </connection>
      <connection>
         <port blockname="CRC.xml" blockid="CRC" name="down" id="down" />
         <port blockname="FlowControl.xml" blockid="FlowCongControl" name="down
             " id="down" />
      </connection>
      <connection>
         <port blockname="FlowControl.xml" blockid="FlowCongControl" name="up"
             id="up" />
         <port blockname="net.xml" blockid="net" name="data" id="data" />
      </connection>
   </connections>
</Workflow>
```

Listing A.4.2: Protocol Graph with AES Encryption

```
<?xml version="1.0" encoding="UTF-8"?>
<Workflow>
   <Optional>
      <Offering>
         <Effect>delay</Effect>
         <Operator>=</Operator>
         <Attribute Unit="">3.0</Attribute>
      </Offering>
      <Offering>
         <Effect>imagequality</Effect>
         <Operator>=</Operator>
         <Attribute Unit="">75.0</Attribute>
      </Offering>
      <Offering>
         <Effect>bandwidth</Effect>
         <Operator>=</Operator>
         <Attribute Unit="">0.0</Attribute>
      </Offering>
      <Offering>
         <Effect>lossratio</Effect>
         <Operator>=</Operator>
         <Attribute Unit="">0.0</Attribute>
      </Offering>
   </Optional>
   <buildingblocks>
      <buildingblock id="app" uuid="app" special="app">
         <Port PortID="data">
            <OfferedEffect Effect="application" Operator="=" Attribute="true"
                />
         </Port>
      </buildingblock>
      <buildingblock id="ImageComp75" uuid="ImageComp75">
         <Port PortID="down">
            <OfferedEffect Effect="DataReduction" Operator="=" Attribute="true"
                />
         </Port>
         <Port PortID="up">
            <OfferedEffect Effect="Data" Operator="=" Attribute="true" />
         </Port>
      </buildingblock>
      <buildingblock id="AES256" uuid="AES256">
         <Port PortID="down">
            <OfferedEffect Effect="ciphering" Operator="=" Attribute="true" />
         </Port>
         <Port PortID="up">
```

```
            <OfferedEffect Effect="Data" Operator="=" Attribute="true" />
         </Port>
      </buildingblock>
      <buildingblock id="Retransmission" uuid="Retransmission">
         <Port PortID="up">
            <OfferedEffect Effect="Data" Operator="=" Attribute="true" />
         </Port>
         <Port PortID="down">
            <OfferedEffect Effect="LossRatio" Operator="=" Attribute="0" />
         </Port>
      </buildingblock>
      <buildingblock id="net" uuid="net" special="net">
         <Port PortID="data">
            <OfferedEffect Effect="net" Operator="=" Attribute="true" />
         </Port>
      </buildingblock>
   </buildingblocks>
   <connections>
      <connection>
         <port blockname="app.xml" blockid="app" name="data" id="data" />
         <port blockname="ImageComp75.xml" blockid="ImageComp75" name="up" id="
            up" />
      </connection>
      <connection>
         <port blockname="ImageComp75.xml" blockid="ImageComp75" name="down" id
            ="down" />
         <port blockname="AES256.xml" blockid="AES256" name="up" id="up" />
      </connection>
      <connection>
         <port blockname="AES256.xml" blockid="AES256" name="down" id="down" />
         <port blockname="Retransmission.xml" blockid="Retransmission" name="up
            " id="up" />
      </connection>
      <connection>
         <port blockname="Retransmission.xml" blockid="Retransmission" name="
            down" id="down" />
         <port blockname="net.xml" blockid="net" name="data" id="data" />
      </connection>
   </connections>
</Workflow>
```

Listing A.4.3: Protocol Graph with Twofish Encryption

```
<?xml version="1.0" encoding="UTF-8"?>
<Workflow>
   <Optional>
      <Offering>
         <Effect>delay</Effect>
         <Operator>=</Operator>
         <Attribute Unit="">4.0</Attribute>
      </Offering>
      <Offering>
         <Effect>imagequality</Effect>
         <Operator>=</Operator>
         <Attribute Unit="">75.0</Attribute>
      </Offering>
      <Offering>
         <Effect>bandwidth</Effect>
         <Operator>=</Operator>
         <Attribute Unit="">0.0</Attribute>
      </Offering>
      <Offering>
         <Effect>lossratio</Effect>
         <Operator>=</Operator>
         <Attribute Unit="">0.0</Attribute>
      </Offering>
   </Optional>
   <buildingblocks>
      <buildingblock id="app" uuid="app" special="app">
         <Port PortID="data">
            <OfferedEffect Effect="application" Operator="=" Attribute="true"
               />
```

```
            </Port>
        </buildingblock>
        <buildingblock id="ImageComp75" uuid="ImageComp75">
            <Port PortID="down">
                <OfferedEffect Effect="DataReduction" Operator="=" Attribute="true"
                    />
            </Port>
            <Port PortID="up">
                <OfferedEffect Effect="Data" Operator="=" Attribute="true" />
            </Port>
        </buildingblock>
        <buildingblock id="Twofish256" uuid="Twofish256">
            <Port PortID="down">
                <OfferedEffect Effect="ciphering" Operator="=" Attribute="true" />
            </Port>
            <Port PortID="up">
                <OfferedEffect Effect="Data" Operator="=" Attribute="true" />
            </Port>
        </buildingblock>
        <buildingblock id="Retransmission" uuid="Retransmission">
            <Port PortID="up">
                <OfferedEffect Effect="Data" Operator="=" Attribute="true" />
            </Port>
            <Port PortID="down">
                <OfferedEffect Effect="LossRatio" Operator="=" Attribute="0" />
            </Port>
        </buildingblock>
        <buildingblock id="net" uuid="net" special="net">
            <Port PortID="data">
                <OfferedEffect Effect="net" Operator="=" Attribute="true" />
            </Port>
        </buildingblock>
    </buildingblocks>
    <connections>
        <connection>
            <port blockname="app.xml" blockid="app" name="data" id="data" />
            <port blockname="ImageComp75.xml" blockid="ImageComp75" name="up" id="
                up" />
        </connection>
        <connection>
            <port blockname="ImageComp75.xml" blockid="ImageComp75" name="down" id
                ="down" />
            <port blockname="Twofish256.xml" blockid="Twofish256" name="up" id="up
                " />
        </connection>
        <connection>
            <port blockname="Twofish256.xml" blockid="Twofish256" name="down" id="
                down" />
            <port blockname="Retransmission.xml" blockid="Retransmission" name="up
                " id="up" />
        </connection>
        <connection>
            <port blockname="Retransmission.xml" blockid="Retransmission" name="
                down" id="down" />
            <port blockname="net.xml" blockid="net" name="data" id="data" />
        </connection>
    </connections>
</Workflow>
```

A.5 Building Blocks

Listing A.5.1: Building Block: Encryption AES256

```
<?xml version="1.0" encoding="UTF-8"?>
<BuildingBlock xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" BBID="
    AES256" xsi:noNamespaceSchemaLocation="BB_Description.xsd">
  <Port PortID="down">
    <Offering>
      <Effect>ciphering</Effect>
      <Operator>=</Operator>
```

```
      <Attribute Unit="ms">
        <Formula Type="value">true</Formula>
      </Attribute>
    </Offering>
  </Port>
  <Port PortID="up">
    <Offering>
      <Effect>Data</Effect>
      <Operator>=</Operator>
      <Attribute Unit="">
        <Formula Type="value">true</Formula>
      </Attribute>
    </Offering>
  </Port>
  <Optional>
    <QOS>
      <Effect>delay</Effect>
      <Operator>=</Operator>
      <Attribute Unit="ms">
        <Formula Type="value">3</Formula>
      </Attribute>
    </QOS>
    <QOS>
      <Effect>lossratio</Effect>
      <Operator>=</Operator>
      <Attribute Unit="%">
        <Formula Type="value">0</Formula>
      </Attribute>
    </QOS>
  </Optional>
</BuildingBlock>
```

Listing A.5.2: Building Block: Checksum (CRC)

```
<?xml version="1.0" encoding="UTF-8"?>
<BuildingBlock xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" BBID="CRC"
     xsi:noNamespaceSchemaLocation="BB_Description.xsd">
  <Port PortID="down">
    <Offering>
      <Effect>Integrity</Effect>
      <Operator>=</Operator>
      <Attribute Unit="ms">
        <Formula Type="value">true</Formula>
      </Attribute>
    </Offering>
  </Port>
  <Port PortID="up">
    <Offering>
      <Effect>Data</Effect>
      <Operator>=</Operator>
      <Attribute Unit="">
        <Formula Type="value">true</Formula>
      </Attribute>
    </Offering>
  </Port>
  <Optional>
    <QOS>
      <Effect>delay</Effect>
      <Operator>=</Operator>
      <Attribute Unit="ms">
        <Formula Type="value">1</Formula>
      </Attribute>
    </QOS>
    <QOS>
      <Effect>lossratio</Effect>
      <Operator>=</Operator>
      <Attribute Unit="%">
        <Formula Type="value">0</Formula>
      </Attribute>
    </QOS>
  </Optional>
</BuildingBlock>
```

Listing A.5.3: Building Block: Flow Control

```
<?xml version="1.0" encoding="UTF-8"?>
<BuildingBlock xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" BBID="
    FlowCongControl" xsi:noNamespaceSchemaLocation="BB_Description.xsd">
  <Port PortID="down">
    <Offering>
      <Effect>Data</Effect>
      <Operator>=</Operator>
      <Attribute Unit="ms">
        <Formula Type="value">true</Formula>
      </Attribute>
    </Offering>
  </Port>
  <Port PortID="up">
    <Offering>
      <Effect>Controlling</Effect>
      <Operator>=</Operator>
      <Attribute Unit="">
        <Formula Type="value">true</Formula>
      </Attribute>
    </Offering>
  </Port>
  <Optional>
    <QOS>
      <Effect>delay</Effect>
      <Operator>=</Operator>
      <Attribute Unit="ms">
        <Formula Type="value">1</Formula>
      </Attribute>
    </QOS>
  </Optional>
</BuildingBlock>
```

Listing A.5.4: Building Block: Retransmission

```
<?xml version="1.0" encoding="UTF-8"?>
<BuildingBlock xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" BBID="
    Retransmission" xsi:noNamespaceSchemaLocation="BB_Description.xsd">
  <Port PortID="up">
    <Offering>
      <Effect>Data</Effect>
      <Operator>=</Operator>
      <Attribute Unit="ms">
        <Formula Type="value">true</Formula>
      </Attribute>
    </Offering>
  </Port>
  <Port PortID="down">
    <Offering>
      <Effect>LossRatio</Effect>
      <Operator>=</Operator>
      <Attribute Unit="%">
        <Formula Type="value">0</Formula>
      </Attribute>
    </Offering>
  </Port>
</BuildingBlock>
```

Listing A.5.5: Building Block: Fragementation

```
<?xml version="1.0" encoding="UTF-8"?>
<BuildingBlock xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" BBID="
    Fragementation" xsi:noNamespaceSchemaLocation="BB_Description.xsd">
  <Port PortID="down">
    <Offering>
      <Effect>Data</Effect>
      <Operator>=</Operator>
      <Attribute Unit="ms">
        <Formula Type="value">true</Formula>
```

```
      </Attribute>
    </Offering>
  </Port>
  <Port PortID="extra">
    <Offering>
      <Effect>Mangement</Effect>
      <Operator>=</Operator>
      <Attribute Unit="ms">
        <Formula Type="value">true</Formula>
      </Attribute>
    </Offering>
  </Port>
  <Port PortID="up">
    <Offering>
      <Effect>Packets</Effect>
      <Operator>=</Operator>
      <Attribute Unit="">
        <Formula Type="value">true</Formula>
      </Attribute>
    </Offering>
  </Port>
  <Optional>
    <QOS>
      <Effect>delay</Effect>
      <Operator>=</Operator>
      <Attribute Unit="ms">
        <Formula Type="value">1</Formula>
      </Attribute>
    </QOS>
  </Optional>
</BuildingBlock>
```

Listing A.5.6: Building Block: Image Compression

```
<?xml version="1.0" encoding="UTF-8"?>
<BuildingBlock xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" BBID="
    ImageComp75" xsi:noNamespaceSchemaLocation="BB_Description.xsd">
  <Port PortID="down">
    <Offering>
      <Effect>DataReduction</Effect>
      <Operator>=</Operator>
      <Attribute Unit="ms">
        <Formula Type="value">true</Formula>
      </Attribute>
    </Offering>
  </Port>
  <Port PortID="up">
    <Offering>
      <Effect>Data</Effect>
      <Operator>=</Operator>
      <Attribute Unit="">
        <Formula Type="value">true</Formula>
      </Attribute>
    </Offering>
  </Port>
  <Optional>
    <QOS>
      <Effect>delay</Effect>
      <Operator>=</Operator>
      <Attribute Unit="ms">
        <Formula Type="value">75</Formula>
      </Attribute>
    </QOS>
    <QOS>
      <Effect>imagequality</Effect>
      <Operator>=</Operator>
      <Attribute Unit="ms">
        <Formula Type="value">75</Formula>
      </Attribute>
    </QOS>
  </Optional>
</BuildingBlock>
```

Listing A.5.7: Building Block: Message Count

```
<?xml version="1.0" encoding="UTF-8"?>
<BuildingBlock xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" BBID="
    MsgCount" xsi:noNamespaceSchemaLocation="BB_Description.xsd">
  <Port PortID="down">
    <Offering>
      <Effect>MessageCount</Effect>
      <Operator>=</Operator>
      <Attribute Unit="ms">
        <Formula Type="value">true</Formula>
      </Attribute>
    </Offering>
  </Port>
  <Port PortID="up">
    <Offering>
      <Effect>Data</Effect>
      <Operator>=</Operator>
      <Attribute Unit="">
        <Formula Type="value">true</Formula>
      </Attribute>
    </Offering>
  </Port>
</BuildingBlock>
```

A.6 Implementation (Java Code)

A Java-based implementation has been carried out as a proof-of-concept. The code is divided into various classes with respect to its scope. Some of the main classes and only few of their attributes and methods are shown in the class diagram in Fig. A.6.1. Where the class ”Composition” is the entrance point. The method ”PerformComposition” accepts application requirements and network constraints as parameters and returns the number of generated protocol graphs. The generated PGs are saved in XML format and ready to be used. The classes ”TemplatePool” and ”BuildingBlockPool” hold all the available templates and building blocks at the host system respectively. The class ”DomainPolicies” holds the policies of all the domains, which is used by the template selection process to choose adequate template(s). The class ”WorkflowGenerator” generates all possible protocol graphs with respect to suitable templates and fitting building blocks for them.

The entire code can be found at Git Repository: `https://github.com/abbas-siddiqui/Template_Based_Composition.git`. The code contains two different ”NetBeans” projects in folders ”Demo_With_GUI” and ”Template_Demo”. The first project runs independently and has a graphical user interface, where a user can load requirements as an XML file. Once the requirements are loaded the GUI will show all available templates in the pool. All available suitable template(s) can be selected by clicking on the button ”Suitable Template(s)”. The button ”Select a Template” will select a single template from the suitable templates, the selection is based on first exact match mechanism. The button ”Fill the Template” can be clicked to generate all possible protocol graphs, these PGs later can be used by the SONATE Framework or other PG execution framework. The

second project does not run independently rather intended to function as a library to be used by other programs or frameworks such as a PG execution framework. Where the class ”Composition” is used as an entry to pass requirements to the composition process and to receive back generated PGs.

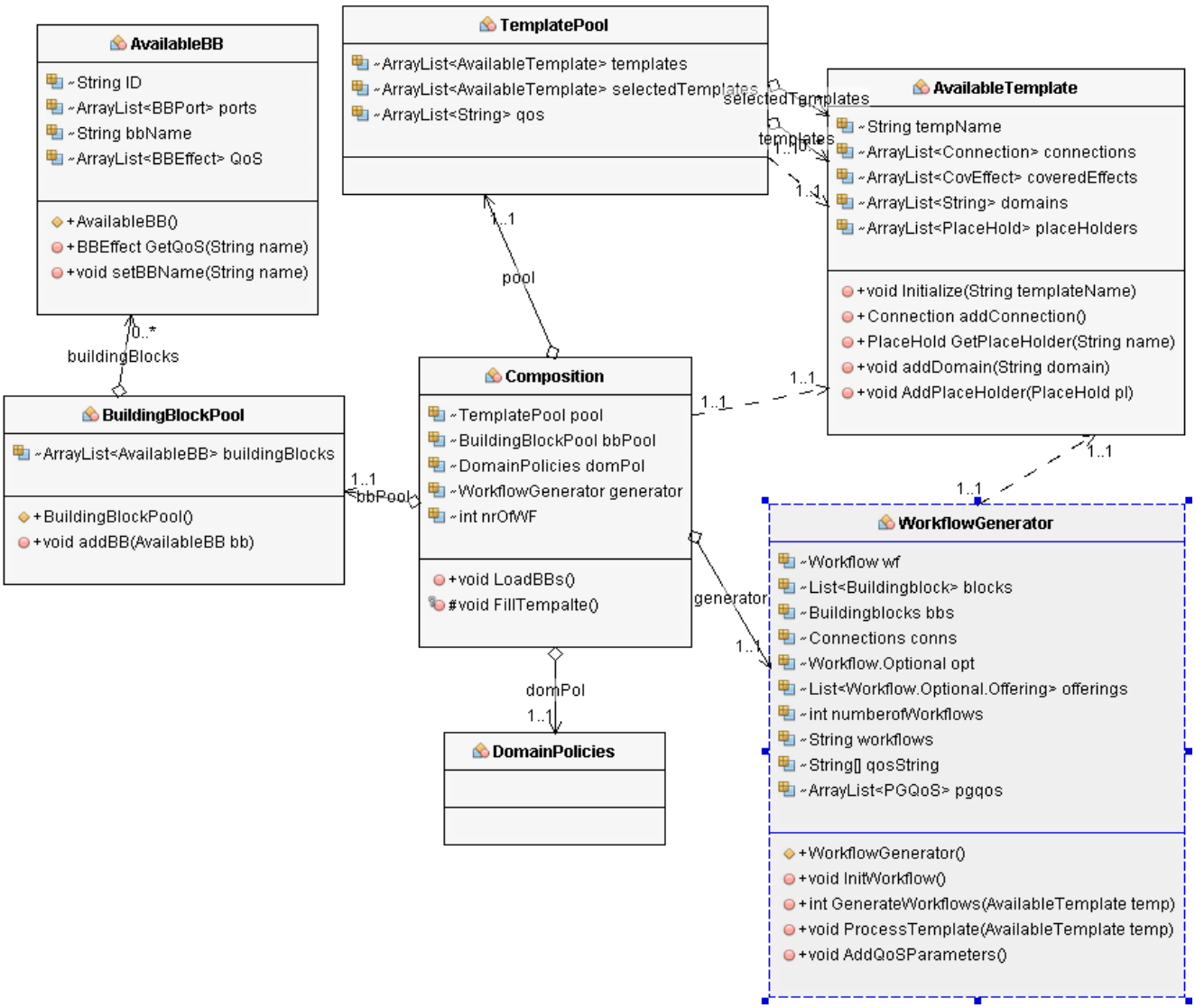


Figure A.6.1: Class Diagram

Bibliography

[AD04] B. Aboba and W. Dixon. Ipsec-network address translation (nat) compatibility requirements. *RFC*, 2004.

[AP10] A. Albreshne and J. Pasquier. Semantic-based semi-automatic web service composition. *Computer Department,Switzerland*, 2010.

[AP11] A. Albreshne and J. Pasquier. A template-based semi-automatic web services composition framework. *European Conference on Web Services ECOWS11*, 2011.

[Bas80] V. R. Basili. Qualitative software complexity models: A summary. in: Tutorial on models and methods for software management and engineering. *IEEE Computer Society Press, Los Alamitos*, 1980.

[BCSW00] R. Braden, D.D. Clark, S. Shenker, and J. Wroclawski. Developing a next-generation internet architecture. *ISI*, 2000.

[BFH03] Robert Braden, Ted Faber, and Mark Handley. From protocol stack to protocol heap: role-based architecture. *SIGCOMM Comput. Commun. Rev.*, 33(1):17–22, 2003.

[BHS+98] Nina T. Bhatti, Matti A. Hiltunen, Richard D. Schlichting, Wanda Chiu, and A Chiu. Coyote: A system for constructing fine-grain configurable communication services, 1998.

[BM05] Fabio Baroncelli and Barbara Martini. A service oriented network architecture suitable for global grid computing. *Optical Network Design and Modeling, 2005. Conference on*, 2005.

[BP06] R. Barrett and C. Pahl. Semi-automatic distribution pattern modeling of web service compositions using semantics. *Enterprise Distributed Object Computing Conference, 2006. EDOC 06*, 2006.

[BY97] Yaneer Bar-Yam. *Dynamics of complex systems*. Studies in nonlinearity. Westview, Boulder (Colo.), 1997.

[CMPK08] A. Carstea, G. Macariu, D. Petcu, and A. Konovalov. Pattern based composition of web services for symbolic computations. *ICCS, 2008*, 2008.

[DRB+07] R. Dutta, G.N. Rouskas, I. Baldine, A. Bragg, and D. Stevenson. The silo architecture for services integration, control, and optimization for the future internet. In *Communications, 2007. ICC '07. IEEE International Conference on*, pages 1899–1904, June 2007.

[EF94] K. Egevang and P. Francis. The ip network address translator (nat). *RFC*, 1994.

[EG03] Matthias Ehrgott and Xavier Gandibleux. Multiple criteria optimization state of the art annotated bibliographic surveys. *Kluwer Academic Publishers*, 2003.

[Erl06] Thomas Erl. *Service-Oriented Architecture*. Prentice Hall, 2006.

[Erl08] Thomas Erl. Soa design patterns. *Prentice Hall 2008*, 2008.

[Fel07] Anja Feldmann. Internet clean-slate design: What and why? *ACM SIGCOMM Computer Communication Review 59 Volume 37, Number 3, July 2007*, 2007.

[FK06] S. Floyd and E. Kohler. Profile for datagram congestion control protocol (dccp) congestion control id 2: Tcp-like congestion control. *RFC4341*, March 2006.

[FK11] S. Frankel and S. Krishnan. Ip security (ipsec) and internet key exchange (ike) document roadmap. *RFC*, 2011.

[FS13] W. Fan and J. Shen. A service composition method based on the template mechanism in the service scalable network framework. *Computer Science and Education (ICCSE), 2013 8th International Conference*, 2013.

[Gea11] Daniel Guenther and et al. A way to identify decision criteria for selecting different mechanisms which provide reliable transmission in a future internet architecture. *7th Conference on Next Generation Internet, EURO-NGI, Kaiserslautern, Germany*, 2011.

[GKS03] Reinhard Gotzhein, Ferhat Khendek, and Philipp Schaible. Micro protocol design: the snmp case study. *SAM'02 Proceedings of the 3rd international conference on Telecommunications and beyond: the broader applicability of SDL and MSC*, 2003.

[GR97] Birgit Geppert and Frank Roessler. Generic engineering of communication protocols - current experience and future issues. In *ICFEM '97: Proceedings of the 1st International Conference on Formal Engineering Methods*, page 70, Washington, DC, USA, 1997. IEEE Computer Society.

[GSM08] K. Geebelen and W. Joosen S. Michiels. Dynamic reconfiguration using template based web service composition. *3rd Workshop on Middleware for Service Oriented Computing, MW4SOC 2008*, 2008.

[GSS+12] Daniel Günther, Dennis Schwerdel, Abbas Siddiqui, M Rahamatullah Khondoker, Bernd Reuther, and Paul Mueller. Selecting and composing requirement aware protocol graphs with sonate. *12th Wurzburg Workshop on IP: ITG Workshop" Visions of Future Generation Networks"(EuroView2012)*, 2012.

[GW07] Sivakumar Ganapathy and Tilman Wolf. Design of a network service architecture. In *Proc. of Sixteenth IEEE International Conference on Computer Communications and Networks (ICCCN)*, Honolulu, HI, August 2007.

[Han06] Mark Handley. Why the internet only just works. *BT Technology Journal*, 24(3), 2006.

[Hea05] Andreas Hanemann and et al. Perfsonar: A service oriented architecture for multi-domain network monitoring. *Springer*, 2005.

[HGW08] Xin Huang, S. Ganapathy, and T. Wolf. A scalable distributed routing protocol for networks with data-path services. In *Network Protocols, 2008. ICNP 2008. IEEE International Conference on*, pages 318–327, Oct. 2008.

[HSee10] Christian Henke, Abbas Siddiqui, and et. el. Network functional composition: State of the art. *Australasian Telecommunication Networks and Applications Conference (ATNAC 2010), Auckland, Newzealand, Oct-2010.*, 2010.

[IEE00] IEEE. 1471-2000 - ieee recommended practice for architectural description of software-intensive systems. *IEEE Standard*, 2000.

[Jea08] William JOHNSTON and et al. Network communication as a service-oriented capability. *High Performance Computing and Grids in Action, Volume 16 Advances in Parallel Computing*, 2008.

[KBGH07] S. Kona, A. Bansal, G. Gupta, and T. Hite. Automatic composition of semanticweb services. *ICWS, 2007*, 2007.

[Kea12] Rahamatullah Khodoker and et al. Usage of analytic hierarchy process for communication service selection. *7th GI/ITG KuVS Workshop on Future Internet 2012, At Nokia Siemens Networks*, 2012.

[KR76] R. L. Kenny and H. Raiffa. Decisions with mulitple objectives: Preferences and value trade-offs. *John Wiley and Sons, New York*, 1976.

[KRS+10] Rahamatullah Khondoker, Bernd Reuther, Dennis Schwerdel, Abbas Siddiqui, and Paul Müller. Describing and selecting communication services in a service oriented network architecture. In *the proceedings of the 2011 ITU-T Kleidoscope event, Beyond the Internet? Innovations for future networks and services, Pune, India*, December 2010.

[KSMB14] M Rahamatullah Khondoker, Abbas Siddiqui, Paul Müller, and Kpatcha Bayarou. Realization of service-orientation paradigm in network architectures. *Journal of ICT*, 2014.

[KSRM12] M. Rahamatullah Khondoker, Abbas Siddiqui, Bernd Reuther, and Paul Mueller. Service orientation paradigm in future network architectures. *Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS-2012)*, 2012.

[KVM11] Rahamatullah Khondoker, Eric MSP Veith, and Paul Müller. A description language for communication services of future network architectures. *In*

Proceedings of the 2011 International Conference on the Network of the Future, pages 69 – 76, 2011.

[KWL+10] A. Kostopoulos, H. Warma, T. Leva, B. Heinrich, A. Ford, and L. Eggert. Towards multipath tcp adoption: Challenges and opportunities. *Next Generation Internet (NGI), 2010 6th EURO-NF Conference on*, 2010.

[LMW+11] Florian Liers, Denis Martin, Hans Wippel, Helge Backhaus, Eric Veith, Abbas Siddiqui, and M Rahamatullah Khondoker. Gapi: a g-lab application-to-network interface. *Euroview 2011, Würzburg*, 2011.

[May72] Jeff Maynard. Modular programming. *London : Butterworths*, 1972.

[MDAD] Steven M.Bellovin, David D.Clark, A.Perrig, and D.Song. A clean-slate design for the next-generation secure internet. `http://www.cs.berkeley.edu/~dawnsong/papers/bellovin_clark_perrig_song_nextGenInternet.pdf`.

[MHK07] A. Jimenez Molina and I. Ko H. Koo. A template-based mechanism for dynamic service composition based on context prediction in ubicomp applications. *International Workshop on Intelligent Web Based Tools (IWBT-07) in conjunction with 19th IEEE ICTAI-07*, 2007.

[MR08] Paul Müller and Bernd Reuther. Future internet architecture - a service oriented approach (future internet architecture - ein serviceorientierter ansatz). *it - Information Technology*, 50(6):383–389, 2008.

[MSBee12] Julius Mueller, Abbas Siddiqui, Martin Becke, and et. el. Evaluating a future internet cross-layer composition prototype. *Testbeds and Research Infrastructure. Development of Networks and Communities*, 2012.

[MSH10] Julius Mueller, Abbas Siddiqui, and Dirk Hoffstadt. Cross-layer security demonstrator for future internet. *3rd Future Internet Symposium 2010 (FIS 2010)*, 2010.

[Mue13] Paul Mueller. Software defined networking: Bridging the gap between distributed-systems and networked-systems research. In Paul Müller, Bernhard Neumair, Helmut Reiser, and Gabi Dreo Rodosek, editors, *6. DFN-Forum Kommunikationstechnologien, Beiträge der Fachtagung, 03.-04. Juni 2013, Erlangen*, volume 217 of *LNI*, pages 43–53. GI, 2013.

[Mun95] G. Munda. Multi criteria evaluation in a fuzzy environment - theory and applications in ecological economics. *Hidelberg: Physika Verlag*, 1995.

[oSC81] Information Sciences Institute University of Southern California. Transmission control protocol. *RFC793*, Sept 1981.

[Pee11] Jianli Pan and et. el. A survey of the research on future internet architectures. *IEEE Communications Magazine July 2011*, 2011.

[Pos80] J. Postel. User datagram protocol. *RFC768*, Aug 1980.

[RD10] Jennifer Rexford and Constantine Dovrolis. Future internet architecture: Clean-slate versus evolutionary research. *communications of the acm vol. 53 september 2010*, 2010.

[Rec94] Recommendation. Recommendation x.200 (07/94), x.200 information technology, open systems interconnection, basic reference model the basic model. *ITU-T*, 1994.

[RH08] Bernd Reuther and Dirk Henrici. A model for service-oriented communication systems. *Journal of Systems Architecture: the EUROMICRO Journal*, 2008.

[RSS+09] Bernd Reuther, Dennis Schwerdel, Abbas Siddiqui, Zornitsa Dimitrova, and Paul Mueller. A protocol framework for a service-orientedfuture internet architecture. *GI/ITG KuVS Fachgespräch Future Internet, Munich*, 2009.

[RSSM09] Bernd Reuther, Abbas Siddiqui, Dennis Schwerdel, and Paul Mueller. An approach towards a flexible network architecture. *Euroview 2009, Würzburg*, 2009.

[RVC01] E. Rosen, A. Viswanathan, and R. Callon. Multiprotocol label switching architecture. *RFC*, 2001.

[Saa80] T. L. Saaty. The analytic hierarchy process. *McGraw-Hill, New York*, 1980.

[Saa08] Thomas L. Saaty. Decision making with the analytic hierarchy process. *Int. J. Services Sciences*, 1(1):83–98, 2008.

[SDS+09] Dennis Schwerdel, Zornitsa Dimitrova, Abbas Siddiqui, Bernd Reuther, and Paul Mueller. Composition of self descriptive protocols for future network architectures. *EuroMicro 2009, 27-29 August, Patras, Greece*, 2009.

[Sea10a] Dennis Schwerdel and et al. On using evolutionary algorithms for solving the functional composition problem. *EuroView2010*, 2010.

[Sea10b] Abbas Siddiqui and et al. Mediation between service and network composition. *10th Würzburg Workshop on IP: Joint ITG and Euro-NF Workshop" Visions of Future Generation Networks"(EuroView2010)*, 2010.

[SGKee11] Dennis Schwerdel, Daniel Guenther, M. Rahamatullah Khondoker, and et. el. A building block interaction model for flexible future internet architectures. *7th EURO-NF CONFERENCE ON NEXT GENERATION INTERNET*, 2011.

[SHP03] E. Sirin, J. Hendler, and B. Parsia. Semi-automatic composition of web services using semantic descriptions. *In Web Services: Modeling, Architecture and Infrastructure workshop in ICEIS*, 2003.

[SHPW09] Shashank Shanbhag, Xin Huang, Santosh Proddatoori, and Tilman Wolf. Automated service composition in next-generation networks. *Distributed Computing Systems Workshops, International Conference on*, 0:245–250, 2009.

[Sif08] Manolis Sifalakis. *Adaptation and Awareness for Autonomic Systems*. PhD thesis, Computer Department Lancester University, October 2008.

[Sip96] Michael Sipser. *Introduction to the Theory of Computation*. International Thomson Publishing, 1st edition, 1996.

[SJMM11] Abbas Siddiqui, Mueller Julius, Becke Martin, and Kleis Michael. Evaluating a future internet cross-layer composition prototype. *7th International ICST Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities, TridentCom, Shanghai, China*, 2011.

[SKM12] Abbas Siddiqui, Rahamatullah Khondoker, and P Muller. Template based composition for requirements based network stacks. *Telecommunication Networks and Applications Conference (ATNAC), 2012 Australasian*, 2012.

[SKR+11] Abbas Siddiqui, Rahamatullah Khondoker, Bernd Reuther, Paul Mueller, Christian Henke, and Helge Backhaus. Functional composition and its challenges. In *The First International Workshop on Future Internet and Next Generation Networks (FIGNet-2011), Seoul, South Korea*, 2011.

[SLee11] M. Sifalakisb, A. Loucaa, and et. el. Functional composition in future networks. *Computer Networks Volume 55, Issue 4, 10 March 2011*, 2011.

[SLJJ+09] X. Sanchez-Loro, J.Casademont, J.Paradells, J. L. Ferrer, and A. Vidal. Proposal of a clean slate network architecture for ubiquitous services provisioning. In *Future Information Networks, 2009. ICFIN 2009. First International Conference on*. IEEE Computer Society, 2009.

[SM11] Abbas Siddiqui and P Muller. Tradeoffs in selection and composition approaches for future internet architectures. *7th GI/ITG KuVS Workshop on Future Internet*, 2011.

[SM12] Abbas Siddiqui and Paul Mueller. A requirement-based socket api for a transition to future internet architectures. *Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS-2012)*, 2012.

[SSCH03] M. Sifalakis, S. Schmid, T. Chart, and D. Hutchison. A generic active service deployment protocol. In *In proceedings of the Second International Workshop on Active Network Technologies and Applications*, pages 100–111, 2003.

[SSSZ94] Douglas C. Schmidt, Burkhard Stiller, Tatsuya Suda, and Martina Zitterbart. Configuring function-based communication protocols for multimedia applications, 1994.

[Sti94] B. Stiller. Fukss: Ein funktionsbasiertes kommunikationssubsystem zur flexiblen konfiguration von kommunikationsprotokollen. *GI/ITG-Fachgruppe Kommunikation und Verteilte Systeme*, 1994.

[STP+11] R. Stewart, M. Tuexen, K. Poon, P. Lei, and V. Yasevich. Sockets api extensions for the stream control transmission protocol (sctp). RFC 6458 (Informational), December 2011.

[SW08] Shashank Shanbhag and Tilman Wolf. Implementation of end-to-end abstractions in a network service architecture. In *CoNEXT '08: Proceedings of the 2008 ACM CoNEXT Conference*, pages 1–12, New York, NY, USA, 2008. ACM.

[SWP+09] Gregor Schaffrath, Christoph Werle, Panagiotis Papadimitriou, Anja Feldmann Rol, Bless Adam, Greenhalgh Andreas Wundsam, Mario Kind, Olaf Maennel, and Laurent Mathy. Network virtualization architecture: Proposal and initial prototype. In *In Proceedings of ACM SIGCOMM VISA*, 2009.

[SXea00] R. Stewart, Q. Xie, and et al. Stream control transmission protocol. *RFC2960*, Oct 2000.

[SZ09] Peter Stuckmann and Rainer Zimmermann. European research on future internet design. *IEEE Wireless Communications Magazine, October 2009*, 2009.

[TWP06] Joseph D. Touch, Yu-Shun Wang, and Venkata Pingali. A recursive network architecture. Online: http://www.isi.edu/touch/pubs/isi-tr-2006-626/, 2006.

[VAMD09] M. H. Valipour, B. Amirzafari, K. N. Maleki, and Negin Daneshpour. A brief survey of software architecture conceptsand service oriented architecture. *Computer Science and Information Technology, 2009. ICCSIT 2009. 2nd IEEE International Conference*, 2009.

[vB11] I. van Beijnum. An ftp application layer gateway (alg) for ipv6-to-ipv4 translation. RFC 6384 (Standards Track), October 2011.

[VME+09] Lars Völker, Denis Martin, Ibtissam El Khayat, Christoph Werle, and Martina Zitterbart. A Node Architecture for 1000 Future Networks. In *Proceedings of the International Workshop on the Network of the Future 2009*, Dresden, Germany, June 2009. IEEE.

[VMW+09] Lars Völker, Denis Martin, Christoph Werle, Martina Zitterbart, and Ibtissam El Khayat. Selecting Concurrent Network Architectures at Runtime. In *Proceedings of the IEEE International Conference on Communications (ICC)*, Dresden, Deutschland, June 2009. IEEE Computer Society.

[VPPW93a] M. Vogt, Th. Plagemann, B. Plattner, and Th. Walter. Eine laufzeitumgebung fuer da capo. *GI/ITG-Arbeitstreffen Verteilte Multimedia-Systeme*, 1993.

[VPPW93b] M. Vogt, Th. Plagemann, B. Plattner, and Th. Walter. A run-time environment for da capo. In *Proceedings of INET93 International Networking Conference of the Internet Society*, 1993.

[VWR+07] Manoj Vellala, Anjing Wang, George Rouskas, Rudra Duttaand Ilia Baldine, and Daniel Stevenson. A composition algorithm for the silo cross-layer optimization service architecture. In *Proc. of the Advanced Networks and Telecommunications Systems Conference (ANTS)*, Mumbai, India, December 2007.

[Wol06] Tilman Wolf. Service-centric end-to-end abstractions in next-generation networks. In *Proc. of Fifteenth IEEE International Conference on Computer Communications and Networks (ICCCN)*, pages 79–86, Arlington, VA, October 2006.

[WOP92] Sean W, O'Malley, and Larry L. Peterson. A dynamic network architecture. *ACM Transactions on Computer Systems*, 10:110–143, 1992.

[ZOP09] E. Zahoor and C. Godart O. Perrin. Rule-based semi automaticweb services composition. *2009 IEEE Congress on Services*, 2009.

Abbas Siddiqui

Curriculum Vitae

✉ *siddiqui.abbas@gmail.com*
Nationality: German

"I have no special talents. I am only passionately curious." - Albert Einstein — To Carl Seelig, his biographer, March 11, 1952. Einstein Archive 39-013

Education

MS in Electrical & Communication Engineering, *University of Kassel*, Germany, *1.1 (Excellent)*.

Bachelor of Engineering (Computer Science), .

Doctorate Thesis

Title *Requirements Aware Template Based Protocol Graphs in SOA Based Network Architecture*

Supervisors Prof. Paul Müller

Description Modularization of a network stack and then compose the desirable service on demand is relatively new in the networks, though, it is being practiced in software engineering since few decades now. Composition of the functionalities to achieve a desired result can be performed at various epochs such as run-time, design-time, deployment-time. However, epochs have trade-offs in terms of the complexity (i.e., required setup time) and the achieved flexibility. In this work, an approach is presented to split the composition in different time-phases (i.e., run-time, deployment-time, and design-time) to provide the flexibility and yet having a practical enough setup time. In this approach, the complex and time consuming activities are performed at less-critical time (i.e., design-time, deployment-time) and rest is done at run-time. The application of the approach is not limited to the SOA network architecture, but the design also considers the needs of Internet-of-Things and Overlay-Networks

Masters Thesis

Title *Implementing SOAP Server & Deploying Web Service Technology based on SOAP for gathered data from Batteries & Capacitors*

Supervisors Prof. Dr.-Ing. Jürgen Schmidt

Description In this work, a software is developed for signaling and data collection from the Batteries & Capacitors. And to publish this collected information on INTERNET or within company's network, a multi-threaded SOAP server is implemented that can serve multiple users simultaneously. The server is independent of the Client implementation as the open standards like SOAP, XML are used for the communication purpose.

Tools & Tech. C, Java, Hardware Cards Library in C, Visual C++ (MFC), UML, J-Builder, XML, SOAP, WSDL, UDDI, and Tomcat Apache

Publications

Software Engineering & Architecture

D. Günther, D. Schwerdel, A. Siddiqui, M. R. Khondoker, B. Reuther, and P. Mueller. Selecting and composing requirement aware protocol graphs with sonate. *12th Wurzburg Workshop on IP: ITG Workshop" Visions of Future Generation Networks"(EuroView2012)*, 2012.

C. Henke, A. Siddiqui, and et. el. Network functional composition: State of the art. *Australasian Telecommunication Networks and Applications Conference (ATNAC 2010), Auckland, Newzealand, Oct-2010.*, 2010.

M. R. Khondoker, A. Siddiqui, P. Müller, and K. Bayarou. Realization of service-orientation paradigm in network architectures. *Journal of ICT*, 2014.

M. R. Khondoker, A. Siddiqui, B. Reuther, and P. Mueller. Service orientation paradigm in future network architectures. *Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS-2012)*, 2012.

R. Khondoker, B. Reuther, D. Schwerdel, A. Siddiqui, and P. Müller. Describing and selecting communication services in a service oriented network architecture. In *the proceedings of the 2011 ITU-T Kleidoscope event, Beyond the Internet? Innovations for future networks and services, Pune, India*, December 2010.

F. Liers, D. Martin, H. Wippel, H. Backhaus, E. Veith, A. Siddiqui, and M. R. Khondoker. Gapi: a g-lab application-to-network interface. *Euroview 2011, Würzburg*, 2011.

B. Reuther, D. Schwerdel, A. Siddiqui, Z. Dimitrova, and P. Mueller. A protocol framework for a service-orientedfuture internet architecture. *GI/ITG KuVS Fachgespräch Future Internet, Munich*, 2009.

B. Reuther, A. Siddiqui, D. Schwerdel, and P. Mueller. An approach towards a flexible network architecture. *Euroview 2009, Würzburg*, 2009.

D. Schwerdel, Z. Dimitrova, A. Siddiqui, B. Reuther, and P. Mueller. Composition of self descriptive protocols for future network architectures. *EuroMicro 2009, 27-29 August, Patras, Greece*, 2009.

A. Siddiqui, R. Khondoker, and P. Muller. Template based composition for requirements based network stacks. *Telecommunication Networks and Applications Conference (ATNAC), 2012 Australasian*, 2012.

A. Siddiqui, R. Khondoker, B. Reuther, P. Mueller, C. Henke, and H. Backhaus. Functional composition and its challenges. In *The First International Workshop on Future Internet and Next Generation Networks (FIGNet-2011), Seoul, South Korea*, 2011.

A. Siddiqui, M. Kleis, J. Mueller, P. Mueller, and T. Magedanz. Application and network services composition with the help of mediation. *11th Würzburg Workshop on IP: Joint ITG and Euro-NF Workshop" Visions of Future Generation Networks"(EuroView2011)*, 2011.

A. Siddiqui and P. Mueller. A requirement-based socket api for a transition to future internet architectures. *Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS-2012)*, 2012.

A. Siddiqui and P. Muller. Tradeoffs in selection and composition approaches for future internet architectures. *7th GI/ITG KuVS Workshop on Future Internet*, 2011.

Adaptability and Context based Filtering in mHealth Architecture

M. Jansen, A. Siddiqui, and O. Koch. Provision of personalized data via mobile web services in ehealth scenarios. *WEBIST 2014*, 2014.

O. Koch, A. Siddiqui, and M. Jansen. Context-based mhealth applications based on mobile web services. *Med-e-Tel 2014*, 2014.

A. Siddiqui, O. Koch, A. Rabie, and U. Handmann. Personalized and adaptable mhealth architecture. *MOBIHEALTH 2014*, 2014.

Future Internet & Security

M. Becke, K. Campowsky, C. H. andJulius Müller, C. Schmoll, A. Siddiqui, and et. el. A demonstrator for cross-layer composition. *10th Würzburg Workshop on IP: Joint ITG and Euro-NF Workshop" Visions of Future Generation Networks"(EuroView2010)*, 2010.

M. Becke, K. Campowsky, C. Henke, A. Siddiqui, and et. el. Addressing security in a cross-layer composition architecture. *10th Würzburg Workshop on IP: Joint ITG and Euro-NF Workshop" Visions of Future Generation Networks"(EuroView2010)*, 2010.

C. Henke, K. Campowsky, A. Siddiqui, and et. el. Scenarios for a future internet based on cross-layer functional composition. *5th GI/ITG KuVS Fachgespräch Future Internet, Stuttgart*, 2010.

M. Kleis, C. Varas, A. Siddiqui, P. Mueller, I. Simsek, M. Becke, and et. el. Cross-layer security and functional composition for a future internet. *Proceedings of 11th Würzburg Workshop on IP: Visions of Future Generation Networks (EuroView2011)*, 2011.

J. Mueller, A. Siddiqui, M. Becke, and et. el. Evaluating a future internet cross-layer composition prototype. *Testbeds and Research Infrastructure. Development of Networks and Communities*, 2012.

J. Mueller, A. Siddiqui, and D. Hoffstadt. Cross-layer security demonstrator for future internet. *3rd Future Internet Symposium 2010 (FIS 2010)*, 2010.

C. Schmoll, C. Henke, D. Hoffstadt, A. A. Siddiqui, and et. el. G-lab deep: Cross-layer composition and security for a flexible future internet. *Testbeds and Research Infrastructures. Development of Networks and Communities*, 2011.

A. Siddiqui, M. Julius, B. Martin, and K. Michael. Evaluating a future internet cross-layer composition prototype. *7th International ICST Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities, TridentCom, Shanghai, China*, 2011.

T. Zseby, C. Schmoll, C. Henke, D. Hoffstadt, and A. Siddiqui. G-lab deep: Cross-layer composition and security for a flexible future internet. *The 6th International ICST Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TridentCom 2010)*, 2010.