VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

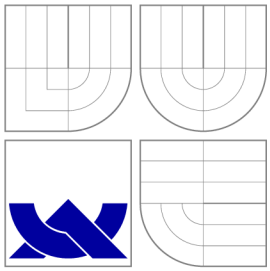
RECURSIVE IPC NETWORK ARCHITECTURE: ANALYSIS AND MODELLING OF ENROLLMENT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

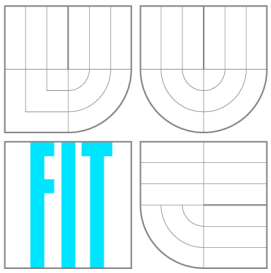
AUTOR PRÁCE
AUTHOR

KAMIL JEŘÁBEK

BRNO 2015



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

REKURZIVNÍ IPC SÍŤOVÁ ARCHITEKTURA: ANALÝZA A MODELOVÁNÍ PŘIPOJENÍ K SÍTI

RECURSIVE IPC NETWORK ARCHITECTURE:

ANALYSIS AND MODELLING OF ENROLLMENT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — KAMIL JEŘÁBEK
AUTHOR

VEDOUCÍ PRÁCE — Ing. PATRIK HALFAR
SUPERVISOR

BRNO 2015

Abstrakt

Tato práce se zabývá analýzou připojení nového člena k síti a začlenením této fáze do modelu Rekurzivní IPC síťové architektury (RINA) vyvíjené v simulačním prostředí OMNeT++. V této práci je obecně popsána architektura RINA. Dále jsou uvedeny možné případy připojení a popsáno kdy tyto fáze začínají a kdy končí.

Abstract

This thesis is focused on analysis of the Enrollment and integration of this phase to model of Recursive InterNetwork Architecture (RINA) that is developed in the OMNeT++ simulation environment. In this thesis is the RINA architecture described generally. Furthermore, there are listed cases of the Enrollment and the Common Application Connection Establishement Phase, and it is described when starts and ends.

Klíčová slova

Simulace sítí, modelování sítí, RINA, OMNeT++, připojení k síti, CACEP, CDAP

Keywords

Simulation of networks, network modeling, RINA, OMNeT++, Enrollment, CACEP, CDAP

Citace

Kamil Jeřábek: Recursive IPC Network Architecture:
Analysis and Modelling of Enrollment, bakalářská práce, Brno, FIT VUT v Brně, 2015

Recursive IPC Network Architecture: Analysis and Modelling of Enrollment

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Patrika Halfara

.....
Kamil Jeřábek
May 19, 2015

Poděkování

Rád bych poděkoval panu Ing. Patriku Halfarovi za ochotu, časovou flexibilitu a za nesčetné rady při zpracovávání této práce. Dále bych chtěl poděkovat kolegům z projektu Pristine, v rámci kterého je RINA simulator vyvíjen, taktéž za významné rady a nápady. V neposlední řadě děkuji své rodině a přítelkyni za psychickou podporu při zpracování mé bakalářské práce.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The Recursive InterNetwork Architecture is new network architecture based on distributed applications. Everything in this architecture could be considered as an application. The Enrollment, which is the main topic of this thesis, is important part of the architecture. Within this phase, the application process is joining to a layer to be a part (member) of it. During it is assigned an address and there are forwarded important information about the connecting layer to a new comming one. In fact, it is the first application communication within a layer that consider if the new member is able to operate within a layer or not and provide this transition. This phase is important because it has to be done for every single application process that want to operate within a layer.

The aim of the present thesis is to create module for the simulation tool OMNeT++ which will allow simulating of Enrollment phase within RINA model. This module will be located as a part of IPC process. The result will be used in the RINA simulator developed in the Pristine project.

1.1 Structure of the thesis

In the following chapter there is a brief description of the Recursive Internetwork Architecture (RINA) provided. There are important notions and the architecture itself named and explained. Chapter 2 covers description of Common Distributed Application Protocol (CDAP) that is important for this thesis. It is important because while establishing application connection as well as while Enrollment are used mainly CDAP messages.

The greatest attention is given to chapters 4 and 5. Each of these chapters covers the description of CDAP messages and its meaning that is used within each phase. Also, there is behavior and detailed design of this phases described.

In the conclusion there are three chapters which describes used environment, the implementation of the model and its validation.

Chapter 2

RINA

In this chapter, there is provided only a brief description of the RINA architecture and its components. These descriptions are necessary to understand other information that this thesis contain. The described components will be also used in whole work.

The Recursive InterNetwork Architecture is distributed network architecture. This architecture is based on the presumption that computer networking is just Inter Process Communication (IPC). The main building block of the architecture is a single repeating layer, the Distributed IPC Facility (DIF). Each occurence of the layer has the same funcions or methods, but from layer to layer it can have different policies. The DIF have a scope and include IPC Processes running on different machines that works together to provide flow services to application processes. The RINA supports without need of creating any extra mobility mechanisms, multihoming and Quality of Service. There is only one protocol, called Common Distributed Application Protocol (CDAP) which providing communication between two application processes.

2.1 Fundamental Definitions

Application Entity (AE) - The task within an application process providing communication with other application processes [8].

Application Process (AP) - The instantiation of a program intended for some purpose, which is executed in system. An AP contains one or more application entities. [8] [9]

Protocol Data Unit (PDU) - The string of octets that two protocol machines exchange among each other. It has two parts. The first one contains information understood and interpreted by the DIF. The second part includes User-Data and what is inside this part of PDU is passed to user. [5]

Service Data Unit (SDU) - The data transferred from one application process to another that are passed across the (N)-DIF interface. The SDU may be fragmented or combined with other SDUs in order to be sent through interface as one or more PDUs. The integrity of SDU is maintained when delivered to a corresponding application protocol machine. [5]

IPC Process - The IPC Process is AP that implements locally funcionality to support and manage IPC. It contains multiple subtasks. We can say that the IPC Process is a DAP

(Distributed Application Process) and it also contains the same components. [6]

Common Distributed Application Protocol (CDAP) - The CDAP is the only required protocol for communication between application processes. The reason, for the only single protocol, is to make a clean transition from the IPC model of the DIF to the programming model of DAFs. There are only six fundamental operations e.g. create/delete, read/write and start/stop. [2] There are not needed more operations for communication. All other maniuplation does not depend on communication, but on manipulation with data in applications. [9]

Common Application Connection Establishement (CACE) - The CACE is a part of CDAP. After the IPC connection is established, the CACE creates an application connection between corresponding peers and initiates authentication. [9]

Distributed Application Facility (DAF) - The DAF consist of two or more application processes on one or more systems that maintain shared state and communicate using IPC. [9]

Distributed IPC Facility (DIF, Layer) - The DIF operates among multiple systems and contains the IPC process running on every of these systems that belongs to the DIF. It is like a blackbox. It means that what happens inside the DIF is not visible outside of the DIF. The DIF is a collection of applications, where each have custom task such as relaying. [2] The DIF is the DAF that provide IPC services to application processes or other DIFs. [8]

(N)-DIF - The actual DIF from which is description based on in hierarchy of DIFs. [8]

(N+1)-DIF - The DIF in hierarchy, which is in N+1 level in view of (N)-DIF. The (N+1)-DIF uses (N)-DIF. The (N)-DIF has information only about applications names of the IPC processes of (N+1)-DIF. The (N+1)-DIF may provide other information to (N)-DIF. [8]

(N-1)-DIF - The DIF in hierarchy, which is in N-1 level in view of (N)-DIF. From this point of view there are applied the same rules between (N)-DIF, which is in N+1 level from (N-1)-DIF. [8]

Resource Information Base (RIB) - Each member of the DIF maintains the RIB, the local representation of the local repository of objects. The distributed application may define a RIB to be its local view of the distributed application. From the perspective of operating systems model the RIB is actually storage. [9]

Relying/Multiplexing Task (RMT) - The RMT is a component of IPC Process that performs multiplexing and/or relying of PDUs. [6]

2.2 IPC Process

The IPC Process is a component of DIF running on a processing system. Every DIF may contain many of IPC processes. On a signle processing system it is intended to be one IPC process that belongs or that operates within one DIF for every DIF that the processing

system is connected to. The IPC process is a DAP (Distributed Application Process) with a specific purpose. To provide IPC services requested by applications. Creating one or more (N)-connections for every Allocate request from applications. And to manage its use of one or more (N-1)-connections with one or more DIFs. The communication within a DIF is allowed only by the IPC Processes. The Figure 2.1 displays, what such IPC Process may include. [6] [8]

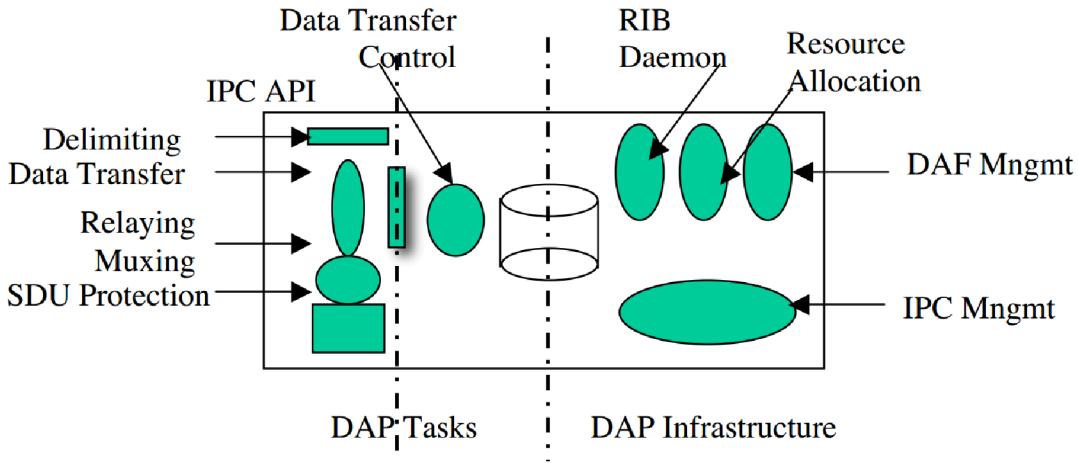


Figure 2.1: IPC Process scheme [6]

2.2.1 The IPC Management

The IPC Management module is component of all DAPs and it is responsible for the use of supporting DIFs (the (N-1)-DIFs through which is going communication with (N)-DIFs), including SDU Protection, multiplexing and the DIF Allocator. The DIF Allocator is used to make a new DIF to a desired application when no one of the available DIFs has access to it. Another part is the RIB Daemon that stores routing tables and other local state information. [6] [10]

2.2.2 Flow Allocator

Flow Allocator is processing all Allocation requests (it is a request to access a requested application). For every single Allocation request is created one separated Instance of the Flow Allocator (FAI). Its function is to find a requested application address of IPC Process that has access to requested application, to determine whether requesting Application Process (AP) has access to requested AP, select policies for flow, monitors and manages the flow. There is also the Name Space Management function which manage the assignment of addresses within a DIF and which is resolving the mapping of addresses/namings from the upper layer to the current DIF addresses/names. This function is also accessed during the Enrollment phase described in 5. [6] [7] [5]

2.2.3 Resource Allocator

Resource Allocator is responsible for managing resource allocations, monitoring and coordinate the IPC Tasks, IPC Management and the Flow Allocator to make sure that quality

of services targets being met and to adjust the policies according to changing conditions. This is performed by sharing information with other DIF IPC Processes. [5] [6]

2.2.4 RIB Daemon

The RIB Daemon is contained in both the DAF and in the DIF. With a little bit different purpose. Especially of the perspective of its content and purpose. The RIB is distributed database that is maintained by every single member of a DIF and a DAF.

For DIF it represents all important logical informations from the point of view of the member such as routing objects important for flows etc. But some informations, as real time information, often used by tasks, will be maintained by them. [6]

The RIB Daemon is responsible for memory management within a DAF. It is common for all subtasks or threads of application process participating in distributed application. Because of every single subtask or thread may require data from the others. It could be asked for a once, more time (it depends on events) or periodically. The RIB Daemon according to the requests may optimize the information. It maintains database synchronization rules e.g. commits on update. [9]

To programming applications the RIB Daemon should be able to provide data within a DAP as it would be on a single system available with possibly no delay. Meant data in application namespaces. Application tasks of DAF members could register subscription to RIB Daemon to take information or to distribute data to one or more members also with defining whether on request or periodically. According to this the RIB Daemon make measurements to reduce the amount of data to be send. [9] [6]

2.3 Addressing and Naming

The naming and addressing is important in any network architecture as well as in RINA. There have to be bound name for every single object. Objects that have no name do not exists. The name is represented by unique string in some alphabet. The name came from given name space, which is a set of N names that are assigned to a collection of objects. A name could be in a time bound to only one object. Synonym designation is when two or more names that are given from one or multiple namespaces that are referencing one object. [8]

There are two main operations for managing names. The Assignment, which allocates name in a namespace after allocation the name is capable to be bound to an object, it is able to use. Deassigment, which remove the name from use. And the Binding, which bind the name to an object that could be accessed through it now. Unbinding breaks the binding and the access to an object through the name is no more be done. [8]

Whatevercast-name is a special identifier taken from namespace that address (gives a reference to) multiple names. In referencing through the name it is used a rule to reference set of names. This is used traditionally for multicast or anycast. [8]

In Table 2.1 are shown the most important parts, which are used by some type of identification and their scope within an architecture. There are need to have identifiers for every single part that are responsible for communication as well as for messages.

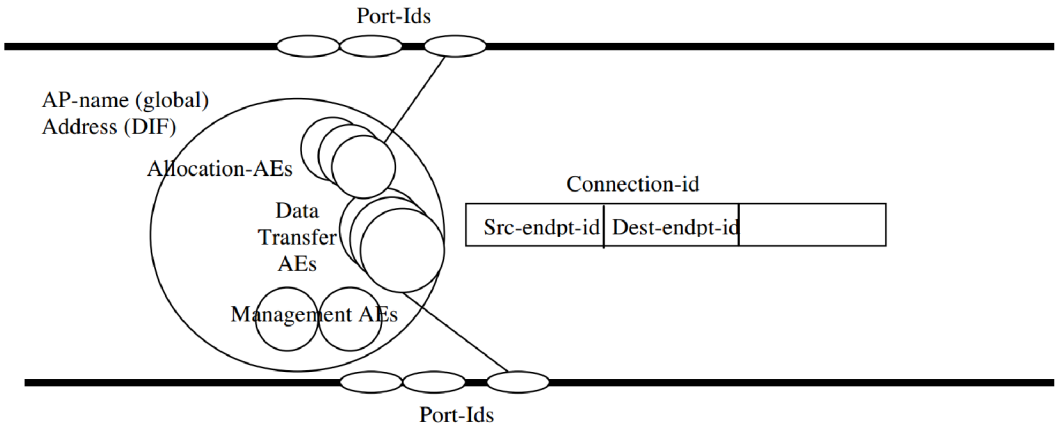


Figure 2.2: The names required for distributed IPC or networking. [6]

Common Term	**Scope**	**Application Term**
Application Process Name	Global (unambiguous)	Application Process Name
Address	DIF (unambiguous)	Synonym for an IPC Process Name
Port-id	Allocation AE of an IPC Process (unique)	AE-instance-identifier
Connection-endpoint-id	Data Transfer AE of an IPC Process (unique)	AE-instance-identifier
Connection-id	Src/Dest Data Transfer AEs (unique)	Concatenation of data transfer AE-instance-identifiers
DIF Management Exchange	IPC Process (unambiguous)	AE-identifier

Table 2.1: Summary of Names required for any Well-Formed Network Architecture [6]

2.4 Communication Protocols

There is necessity to submit that there is a difference between application, which modify shared state external to protocol itself and data transfer protocols that have no external effects than delivering SDUs. [9]

Data Transfer Protocols - Could be devided into two phases. The Allocation phase, which has management function rather than data transfer function. It finds the requested application and determines whether the requesting application has access to it. Then determines whether the requested application could be supported and allocate resources to support IPC. And the Data Transfer Phase, which is responsible for ensuring that the requested properties for communication are provided. [9]

Application Protocols - Also could be devided into two phases. The Initiation phase, which ensure that the two applications are communicating with who they think they should be, i.e. authentication. And an Operations phase, in which operations do modifications

external to the protocol. [9]

2.4.1 Common Distributed Application Protocol

The Common Distributed Application Protocol (CDAP) is the only required protocol for communication between application processes. This protocol well be described in more details in chapter 3.

2.4.2 The Error and Flow Control Protocol

The Error and Flow Control Protocol (EFCP) on each allocation request provides IPC connection/flow. After the Flow Allocator Instance return that it was successfull the EFCP connections are bound to two correspondents AEs. There could be a wide range of protocol to do that. Communication proceeeds only between two EFCP machines. [6]

There are determined three timers: Maximum Packet Lifetime (MPL), Maximum Delay on Ack (A), and Time to Complete Maximum Retries (R). This is used to help to controll communication e.g. when a traffic was not indicated for 2(MPL + A + R) the state between Error and Flow Control Protocol machines could be discarted. [6]

2.4.3 DTP

It is a loosely bound protocol. When there is need for retransmission or flow control for an Allocation request, this protocol is instantiated. There is a state vector used to coordination of protocol machines. [6]

Chapter 3

Common Distributed Application Protocol

This chapter provides only basic description of Common Distirbuted Application Protocol (CDAP) and the description of information that different messages may contain. There is no need to pay attention to significance of each message. Meaning and usage of CDAP messages will be described in two sections later. However, it will describe only messages that are important and used within phases that this thesis deals with.

Such an attention is layed to this protocol in this thesis because the Common Application Connection Phase (CACEP) and the Enrollment, which are the main themes of this work, is using mainly its messages for communication.

3.1 The CDAP overview

The CDAP is the only required protocol for the communication among application processes. The reason for the only single protocol is to make a clean transition from the IPC model of the DIF to the programming model of DAFs. From the point of view of the application there are only six fundamental functions that application can perform on objects. These operations are **create/delete**, **read/write** and **start/stop**. The CDAP is in basically composed from three modules as it is seen in Figure 3.1. [9]

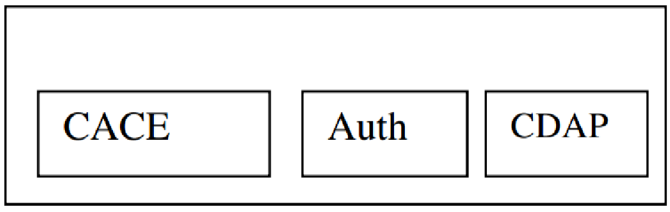


Figure 3.1: The CDAP module scheme [9]

The first one is the common application connection establishement module (CACE). This module which is a part of CDAP module is based on a simplified version of the OSI ACSE protocol. Its behavior and use will be described in more detail in chapter 4. The next is the authentication module and the last one is CDAP. [9]

The modules shown in Figure 3.1 are orientation modules specifying a phase or a part of the communication. The CACE part presents the first exchange of the communication between two correspondents.

The authentication is not strictly defined. There could be many authentication modules for every single application. There is also possibility to use or create own protocol. Each authentication within different types of connections could be in the range from using only password, encryption etc. The authentication part could be entirely skipped. [9]

3.2 Format and type of messages

CDAP messages contain limited types of information. Not all of them are entirely used in every type of message. Table 3.1 displays information held by each different CDAP message.

GPB name	M_CONNECT	M_CONNECT_R	M_RELEASE	M_RELEASE_R	M_CREATE	M_CREATE_R	M_DELETE	M_DELETE_R	M_READ	M_READ_R	M_CANCELREAD	M_CANCELREAD_R	M_WRITE	M_WRITE_R	M_START	M_START_R	M_STOP	M_STOP_R
absSyntax	M	M																
authMech	A	A																
authValue	A	A																
destAEInst	A	A																
destAEName	M	V																
destApInst	A	A																
destApName	M	V																
filter					A	V	A	V	A	V			A	V	A	V	A	V
flags	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C
invokeID	M	=	A	=	A	=	A	=	A	=	=	=	A	=	A	=	A	=
objClass					A	V	A	V	A	V			A	V	M	V	M	V
objInst					M	V	M	V	M	V					M		M	
objName					M	V	M	V	M	V					M		M	
objValue					A	A				C			M	A	A	A	A	V
opCode	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M
resultReason		C		C		C		C		C	C	C		C		C		C
result		M		M		M		M		M	M	M		M		M		M
scope					A		A		A				A		A		A	
srcAEInst	A	A																
srcAEName	M	V																
srcApInst	A	A																
srcApName	M	V																
version	M	M	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A

Table 3.1: CDAP message field table [3]

According to markers in the fields that show which information is carried by messages in which case. The meaning of markers are explained in Table 3.2.

Symbol	Explanation
M	Mandatory, must be present in a PDU of this type
A	Present at application option, not required or validated by CACE
V	Presence is an application option, but the value may be validated against the expected default by CACE for error resilience and reservation for future semantic changes
C	Conditional, may be required by CACE, as described in the PDU operation.
=	Value mandatory, must match that of invokeID in associated PDU
(blank)	Not present, presence is considered an error

Table 3.2: Legend for CDAP message field table [3]

There are several information fields that could be grouped by its significance. The one is the source and destination information. The source as well as destination information each are represented by four fields. That clearly identifies the correspondent and the receiver of information on the level as is needed. **Source Application-Entity-Instance-Id** (srcAEInst in table, string type) identifies the exact instance of the AE originating the message. **Source Application-Entity-Name** (srcAEName, string) is a name of AE within the application. **Source Application-Process-Id** (srcApInst, string) is instance of application originating the message. **Source Application-Process-Name** (srcApName, string) is name Application originating the message. The destination information is caried also in such fields but specifying the destination. [12]

The another one is intended to identificate and transition of objects inside messages. **ObjectClass** (objClass, string) identifies an object class definition of the addressed object. **ObjectInstance** (objInst, int64) carries information that identifies the single object with its specific object class and object name in applications RIB. **ObjectName** (objName, string) is an unique identifier (the name) of the specific object of the specific object class that distinguish the object within application. **ObjectValue** (objValue, Message) may contain scalar, array or compound object type and value.[3]

Furthermore, there is authentication which is a essential part of the connection messages providing first simple authentication of the communication. This needs only two fields. **Authentication-MechanismName** (authMech, string) presenting the identification of the authentication method that destination application use to authenticate the source application. The second one is **Authentication-Value** (authValue, bytes) carrying a format and a value appropriate to the used authentication mechanism. [3]

The last group is reserved for the result. It presents also two fields. **Result-Reason** (resultReason, string) which provides the addition explanation of **Result** (if it is necessary). The **Result** (result, (enum) sint32) that represents result of operation indicating the degree of success or failure of the requested operation. [3]

Moreover, there are other fields that could not be grouped such as that above. These types will be described separately. **AbstractSyntaxID** (absSyntax, int32) determines specific version of the CDAP protocol message declarations that the message conforms to. **Opcode** (opCode, (enum) int32) carries the type of the CDAP message type. [3]

InvokeID (invokeID, int32) is an unique identifier provided to identify request and other messages associated with it. There is also a possibility to have no invokeID. In this

case the invokeID is represented by value 0. If a request contains no invokeID, the response is not required. It is provided either by the application or the AE. This is an integer value that could be taken from a pool of different size of space of numbers (applications could cycle through that amount of messages as it is needed). If the invokeID is provided in a message, CDAP creates a transaction state machine. It checks all responses against all active invokeID's. If no match is found, the message is discarded. In case that a match is found, the message is processed appropriately to the application. [3]

When CDAP is used in an environment where the application manages a hierarchy of objects. There is a place to use a field **Scope** (scope, (enum) int32) that determines on which level in the hierarchy the given operation should be done. In addition, **Filter** (filter, bytes) presents an interpreted predicate function. This function determine whether the operation should be applied on a specific object. [3]

Version (version, int32) specifies a version of RIB as well as a version of an object set to use in the conversation. The last one is **Flags** (flags, (enum) int32) modify meaning of a message in a uniform way when true. [3]

Chapter 4

Common Application Connection Establishment Phase

This chapter contains the analysis and the description of the application connection phase. The emphasis is on the CDAP protocol messages used within this phase, the description of the begining and the end as well as on the overall description.

4.1 Type of messages

GPB name	M_CONNECT	M_CONNECT_R	M_RELEASE	M_RELEASE_R
absSyntax	M	M		
authMech	A	A		
authValue	A	A		
destAEInst	A	A		
destAEName	M	V		
destApInst	A	A		
destApName	M	V		
flags	C	C	C	C
invokeID	M	=	A	=
opCode	M	M	M	M
resultReason		C		C
result		M		M
srcAEInst	A	A		
srcAEName	M	V		
srcApInst	A	A		
srcApName	M	V		
version	M	M	A	A

Table 4.1: CDAP message field table of messages used within CACEP [12]

To CACE phase could be assigned four CDAP messages. Two messages `M_CONNECT`

and M_CONNECT_R response are responsible to create the application connection between two correspondents. In the following table 4.1 there should be seen what these CDAP messages contain. The more specific description of CDAP messages is provided in chapter 3.

While establishing the application connection there has to be determined not only which specific version of CDAP will be used during communication but also the version of RIB and objects. These messages also carry the initial authentication. There is need to use invokeID because the response is required. These two messages are also the only CDAP messages that contain the information specifying source and the destination peers. Whether the connection should be established matters on what respondent inserts into the Result. [12]

The releasing of the connection is also associated with CACEP. Two messages are responsible for that: the M_RELEASE and M_RELEASE_R. Those messages contain only the minimum of the information. InvokeID in M_RELEASE request is optional. This depends on the application and on the case why the request is sent. Response is also carried by M_RELEASE_R as well as in all the other responses. [12]

4.2 Description of behavior

At least two a bit different cases could be distinguished in RINA when CACEP is taking place. The first one is to establish the application connection between two IPC processes (between initator and member of DIF) e.g. management flow. The second one is to establish the application connection between AEs within DAF. The difference is especially between the start of DIF and DAF application connection.

4.2.1 The CACE within DIF

IPC process is instructed to join a DIF by receiving allocate request from the Application Process of the DAF, specifying the application process name and the Quality of Service parameters it requires.

At first, the initing IPC process (initiator) allocates the communication with a member of the DIF. This is an application-to-application connection. This connection is used for the internal DIF management and it is the first connection that is created with the member IPC process. After the successful allocation of the management flow the initiator starts with CACEP. It is the first communication, over the successfully allocated flow, which is await by the other side of the communication (responder/member IPC process). In addition, right after initial connect request/response there is a possibility for the other authentication exchanges as requested. If other authentication techniques are used depends on policy of a DIF. [9]

When CACEP succeds the CDAP connection is established and the Enrollment may proceed. After the successfull Enrollment the initiator have assigned an address. The initiator could now allocate a data transfer connection with application processes of the appropriate DAF that uses the DIF.

4.2.2 The CACE within DAF

This phase may proceed within the DAF after the IPC process has assigned an address, when it is authenticated member of the DIF. The CACEP occurs nearly after the initiating application process obtain a positive allocation response for the requested connection.

The corresponding AEs exchange initial CACE request response. This CACEP should be more application specific then in the DIF. After the initial CACE request/response can take place additional authentication procedures as needed. [9]

The primary function of creating this application connection is to authenticate each side of the communication. And also to establish the set of objects that remote operations on the created flow have access to. [9]

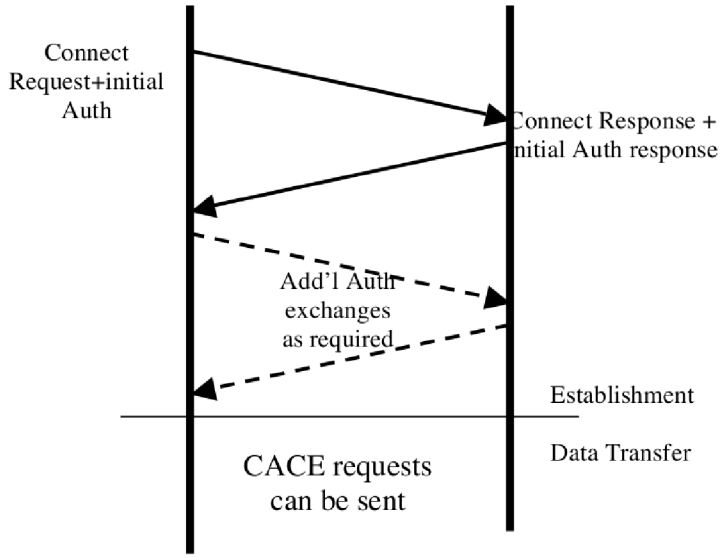


Figure 4.1: Connection establishment over the newly allocated flow [9]

The picture 4.1 shows what was described earlier. At first, it has to be established connection with the IPC process. Then could be send other CACE pdu's to the application. [9]

4.2.3 Release

The connection release belongs to the CACEP as well. The releasing connection depends if it is on demand or as a reaction of unexpected behavior. The release could be initiated by both sides in every state of the communication. [12]

If is it on demand, it is mostly caused by the application for example the application send/receive all the data it wants and it is closing the connection now. In this case the release could be shortened or not. All depends whetеher the application requires the release response or not. [12]

If it requires it sends the CDAP release message with invoke ID provided in the message and wait, for the release response. When receiving the release response the application process may have an option to leave this connection/flow open and reuse it later. If it not requires the release response, the application process sends the CDAP release message without invoke ID and immediately deallocates flow. [12] [3]

If it is the reaction on an unexpected behavior like received unexpected type of the CDAP message in the current state of communication. The receiver of this message sends a release and immediately deallocate the flow. Because this could be a suspicious behavior that can indicate an attack. [12] [12]

4.3 Requirement for CACE

There is a requirement for CACE from a security perspective. To establish every IPC connection there is at least the CACE exchange expected. When it is known what the first message should be and it is expected. It prevents from some attacks of third parties. [6]

4.4 Draft of CACEP

The common application connection establishement phase starts right after it receive positive allocation. More information about when CACEP starts is provided in section 4.2.

The following two state diagrams were designed according to known information and specifications. This is the uniform draft for every connection. Communication always takes place between two endpoints, the initiating and the responding process. There is a small difference between them.

After receiving the positive allocate response the initiator sends `M_CONNECT` message with the appropriate authentication and others required values included. When application receive `M_CONNECT` request it validates the information included in the message as well as authentication values. If the responding process receives a valid message with a valid authentication then it sends `M_CONNECT_R` response with a positive result and transition to the established state. If it is the case, in which the Enrollment is followed by. The member will wait for `M_START Enrollment`. Otherwise the connection is established and the data transfer phase may proceed.

On the other hand, if the message is not valid or the authentication failed, the responding process sends `M_CONNECT_R` response with the negative result. There does not need to be more information about which failure occures, that could help attackers. After the responding process sends negative result it increases a number of connection retries and set

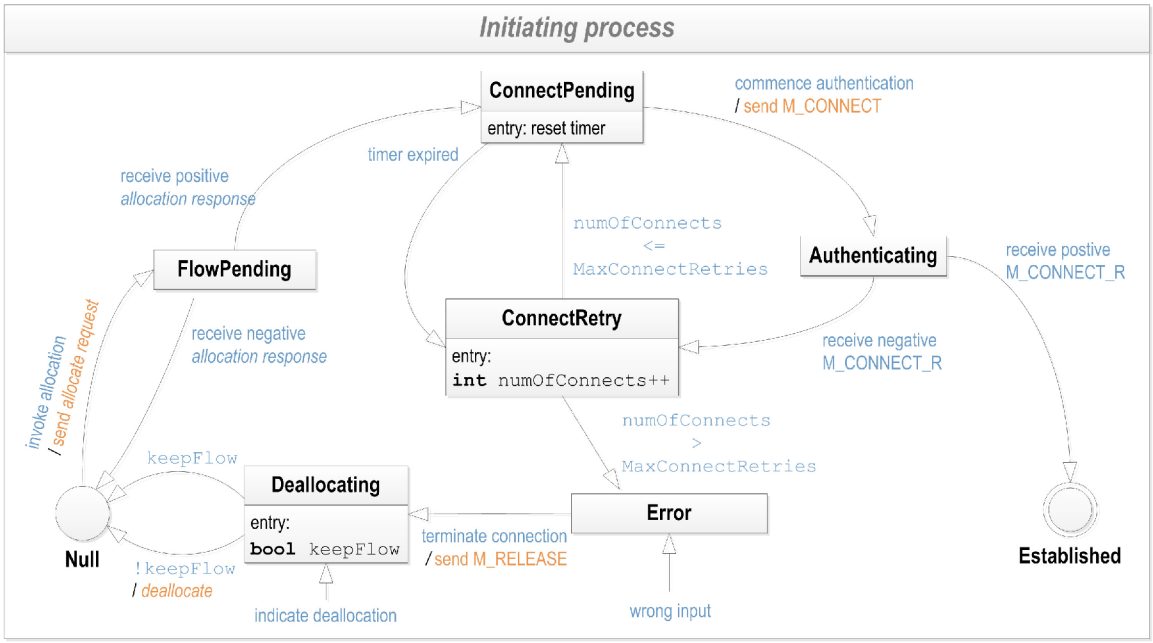


Figure 4.2: Initiating process CACE State Diagram

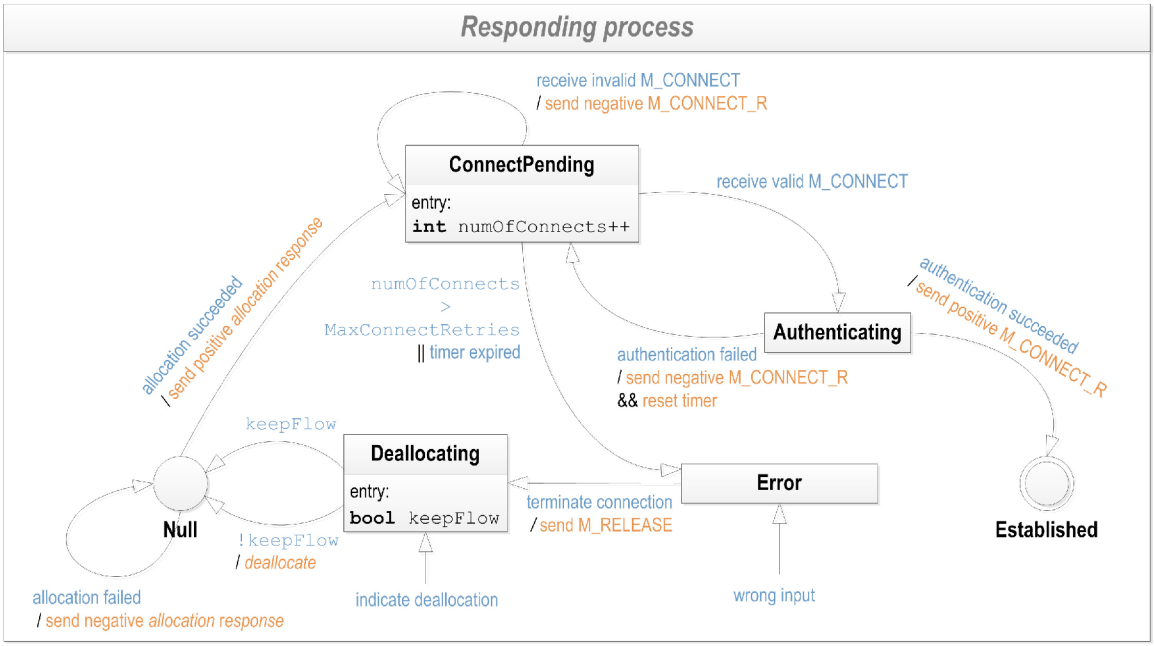


Figure 4.3: Responding process CACE State Diagram

a timer. If the timer expires or max connection retries are reached it sends `M_RELEASE` with no response requested and deallocates all associated with this connection.

When the initiating process receives a negative `M_CONNECT_R` response it could create new `M_CONNECT` request with repaired information inside, or it could send `M_RELEASE` with no response requested and immediately deallocates. If the `M_CONNECT_R` response is positive, the initiating process transition to established state and if it is that case when the Enrollment is followed, it sends `M_START Enrollment` so the Enrollment starts. Otherwise the connection is established and the initiating and responding process may start to communicate with each other.

The last state diagram 4.4 shows releasing. Either the initiating or the responding process may in any state send `M_RELEASE`. This may occure if it receive unexpected message in any state, if the timer expired or if max connection retries are received (as was described earlier). In that cases no response is required and all is deallocated. There is also case in which this is on demand from the application process itself. There should be the `M_RELEASE_R` response expected. There sould be a case when the application want to reuse the flow later. So after receive the `M_RELEASE_R` response the initiator keep the flow open and uses it after again. [3] Otherwise deallocates all associated with the connection.

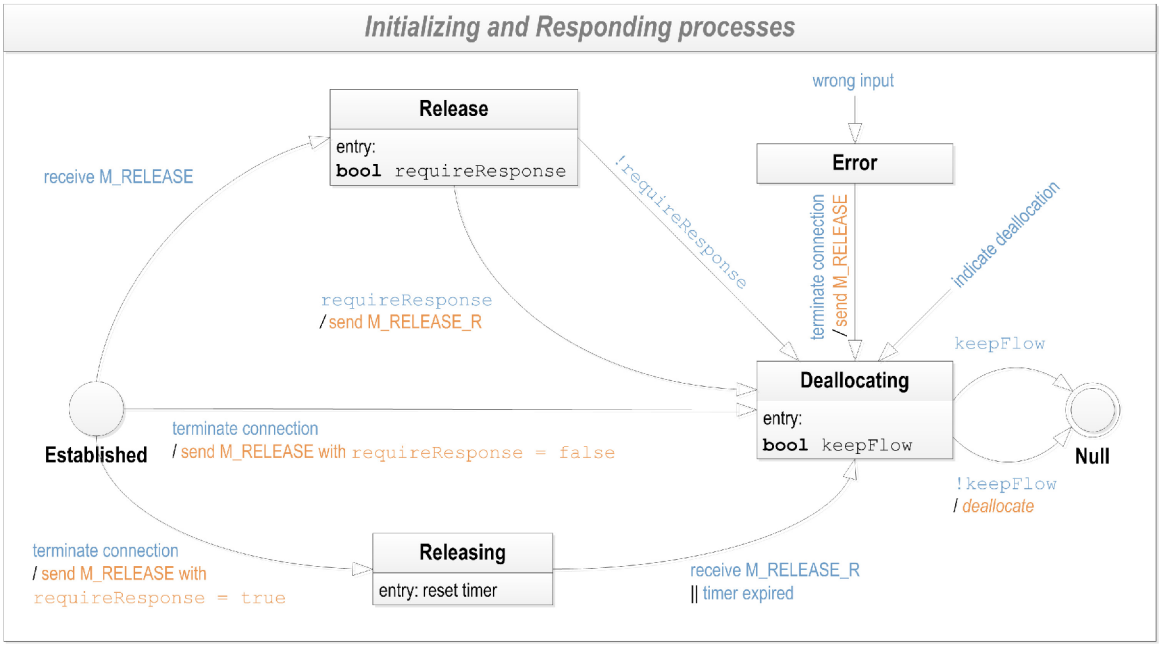


Figure 4.4: Both processes Release CACE State Diagram

Chapter 5

Enrollment

This chapter describes basic concepts of Enrollment. Furthermore, there will be portrayed in more detail design of the Enrollment.

5.1 Type of messages

In the course of the Enrollment are used many of CDAP messages. All these messages are listed in the following table.

GPB name	M_CREATE	M_CREATE_R	M_READ	M_READ_R	M_CREATE	M_CREATE_R
filter	A	V	A	V	A	V
flags	C	C	C	C	C	C
invokeID	A	=	A	=	A	=
objClass	A	V	A	V	A	V
objInst	M	V	M	V	M	V
objName	M	V	M	V	M	V
objValue	A	A		C	A	A
opCode	M	M	M	M	M	M
resultReason		C		C		C
result		M		M		M
scope	A		A		A	
version	A	A	A	A	A	A

Table 5.1: CDAP message field table of messages used within Enrollment only. It is a part of CDAP field table 3.1. Legend represented by table 3.2 belongs to this table. The empty fields were removed. [3]

All of the messages are focused on handling with objects. As was said in chapter 3 applications can only perform three operations and its oppositions such as **create/delete**, **read/write**, and **start/stop**. Within Enrollment there are used only **create**, **read** and **start/stop** operations.

The first messages used is `M_START` with its `M_START_R` response. With start request begin process of making an object operational. According to start there are also `M_STOP` with its response `M_STOP_R`. As start request makes object operational, the stop put an end to operational state above that object. Both requests and responses may contain additional information in the form of objValue.

Another messages are `M_CREATE` with its `M_CREATE_R` response. This request have to contain appropriate values that the object could be created. Except needed fields such as objInstance and objName there could be transmitted values to be bound to given object.

Last messages used within Enrollment are `M_READ` and its `M_READ_R` response that serve to read additional information. There is result and eventually requested values carried in response. [3]

5.2 Basic Concepts

Enrollment is the first phase that all communication must go through. The purpose of Enrollment is to create sufficient shared state that an instance of IPC can be created.

5.2.1 Creating a New DIF

From the point of view of (N)-DIF. It creates initial IPC Process and then connect it to one or more (N-1)-DIFs. This created IPC Process could be directed to start Enrollment with other IPC processes or could wait for other IPC Process that want to enroll with it. [11]

Actually the creation of a New DIF does not fall under the Enrollment phase. But it is one of the important part and the Enrollment will be needed as soon as after the creation of a DIF.

5.2.2 Connecting to a DIF

Enrollment starts after a CDAP connection is established. Firstly, IPC process must allocate communication with a member of the DIF it is instructed to join. The IPC process do this by using (N-1)-DIF that provide sercices both to the IPC process and to one or more IPC processes that are members of the DIF, which the IPC process is trying to join. This connection is used for internal DIF management.

After initial CACE request/response and when the both processes have authenticated each other, the Enrollment may proceed. While authenticating member IPC process determine if the IPC process is allowed to join the DIF or not. Entire CACE phase is described in more deatil in chapter 4.

If it is allowed to join, than member IPC process may request the IPC process for list of (N-1)-DIF-names the IPC process has access to. It could be useful to determine the location of the IPC process network.

This information should be also notifed by the IPC process. The member IPC process or other member IPC processes may also have access to these (N-1)-DIFs, wich could be used as supporting DIFs, if the IPC process wish to support multiple paths within the DIF.

The member of the DIF assigns a synonym Application-process-name to a newcomers IPC process. (In fact, the Name Space Management function, which is incorporated in Flow Allocator, is accessed during the Enrollment to obtain a synonym (address) for use internal to the DIF. The policy of the NSM may also delegate blocks of adresses to members

of the DIF [6].) This synonym could be understand as address, which is used to internal coordination. It has scope only within the DIF and could be structured to facilitate its using within the DIF. In addition to the address, to the IPC process may be given a certificate that can be used to identify itself to other members of the DIF. The two corespondents can exchange some addition relatively static information used by the DIF, such as what policies are in place (range, curently in place), supporting DIFs, etc. There will be also RIB update.

After that, when RIB update is complete, the IPC process is a full-fledged member of the DIF.

We can distinguish two possible types of joining to a DIF. A Naive Case, where an IPC process is instructed to join an existing DIF, which is described above. A Pragmatic Case, where is presupposed that a member lost contact with the DIF due to crash or failure of the physical media.

In this case, Enrollment should be as quick as possible, and that a potentially "new member" not need to be completely initialized.

The initialization starts as above, the "new member" allocates connection to the DIF member, initial CACE request/response and then authenticate the "new member". The „new member" might use the certification to shorten the authentication process.

The IPC process send address and other information to member IPC process. If the address is NULL or assignation expired the member automaticly assign new address. By this, the Enrollment is complete. If the IPC process has mulple (N-1)-DIFs that is also used by members of this DIF, than flows are created with them. [7][5][4]

5.3 Draft of Enrollment

From the perspective of implementation we can clasify the Enrollment as a finite state machine or a part of finite state machine. On the one side is always initiator a process requesting to become member of a DIF. On the oposite side is target a process that is a member of the DIF. There are several CDAP messages used during the Enrollment phase. There is no strict sequence that has to be observed. This fact is given mainly because the initiator may or may not read any addition information not provided by the existing member. That means that the initiator may only read information about whole DIF it wants to join. It is expected that there could be at least two situations for the Enrollment phase. First one, when a process was member of a DIF and now is reconnecting or is still member of the DIF and enroll to neighbor member of the DIF (multiple paths). Second one, when a process is trying to connect for the first time. [5] [7]

The actual Enrollment starts when the initiator recieves `M_CONNECT_R` response. Then the initiator send `M_START Enrollment` containing *Address* which could be *Null* (not member before) or *Address* (if rejoining), *Address_Expiration* and other useful information. Member send `M_START_R` response with *Address, Application_Process_Name, Current_Address, Address_Expiration* included.

The RIB Deamon of the member processes the *M_Create* messages that comprises informations such as *Neighbours* and *Directory forwarding table entries*. This is followed by the `M_CREATE_R` response carrying result of create request. Nearly after the member sends all information in `M_CREATE` messages, it sends `M_STOP Enrollment` containing Boolean value. If the value is true, the initiator will be free to transition to the Operational state. On the other hand, if the value is false, than the initiator cannot transition to the

Operational state until an `M_START Operation` is recieved. However, the initiator may read another information from the member by sending `M_READ` containing number from zero specifying object that want to read. After the initiator finished reading, it sends the `M_STOP_R Enrollment` response.

The member sends `M_START Operation` message and New Member reply with `M_START_R Operation` response. By this the Enrollment is done and operation within DIF started.

5.3.1 Become the New Member of a DIF

The New Member was not connected to this DIF yet. That means that it has no information about the DIF. The Enrollment could be more extensive in this case. This scheme displays

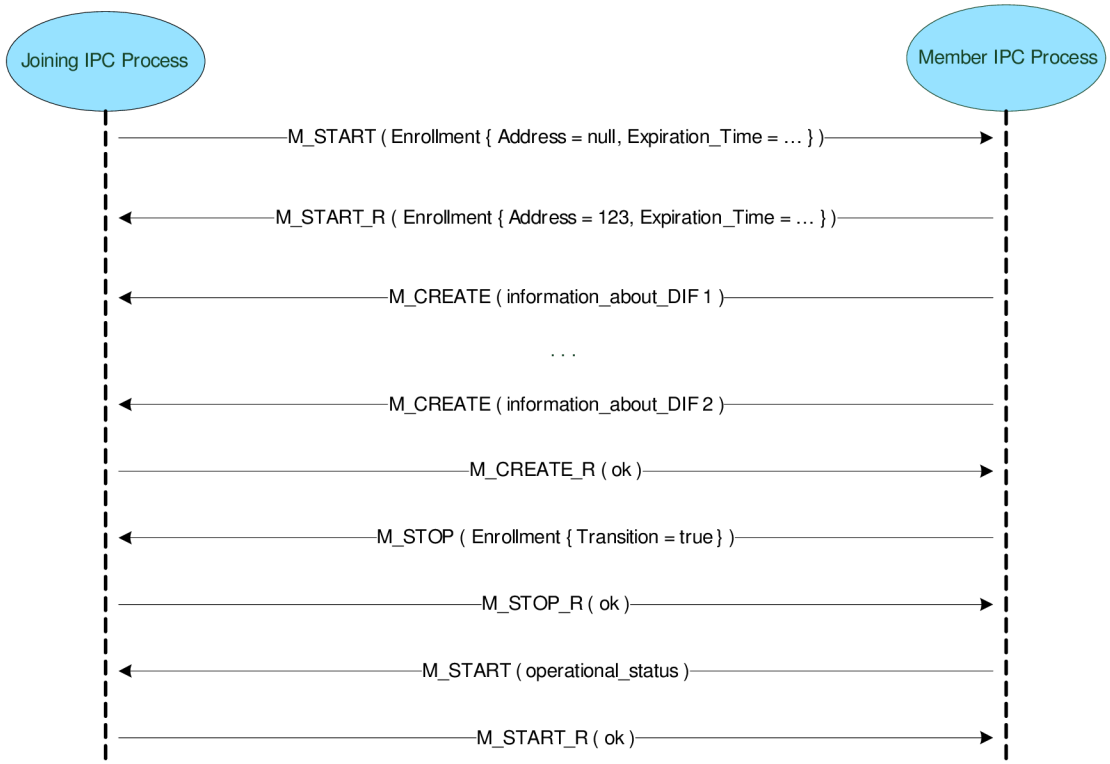


Figure 5.1: Enrollment communication when IPC Process is joining a DIF for the first time

whole Enrollment phase, starting after the new member received `M_CONNECT_R` response from the member. It starts with sending the `M_START Enrollment` message containing *address* = *null* (not member before). Depends on the location within a DIF the member sends `M_START_R Enrollment` response with address for the new member and also other informations such as *Address_expiration*.

Than the member sends series of `M_CREATE` messages followed by the `M_STOP Enrollment` with the positive Boolean value. In this case, the new member do not read any information. Now, when the new member have a DIF in common with one or more of the neighbors, this information recieves through the `M_CREATE` messages (it is multihomed).

The member sends `M_START Operation` and the new member is successfully enrolled.

5.3.2 The Enrollment to known DIF

In this case the New Member (initiator) was a member of a DIF, but according to momentary loss or failure on physical media, it lost a connection with the DIF. The initiator may still have the old address that it had before.

The another possibility is that the initiator is a member of the DIF and is trying to enroll to another member through different (N-1)-DIF for supporting multipath connection to the DIF. As is shown in this scheme, the initiator sends `M_START Enrollment` request

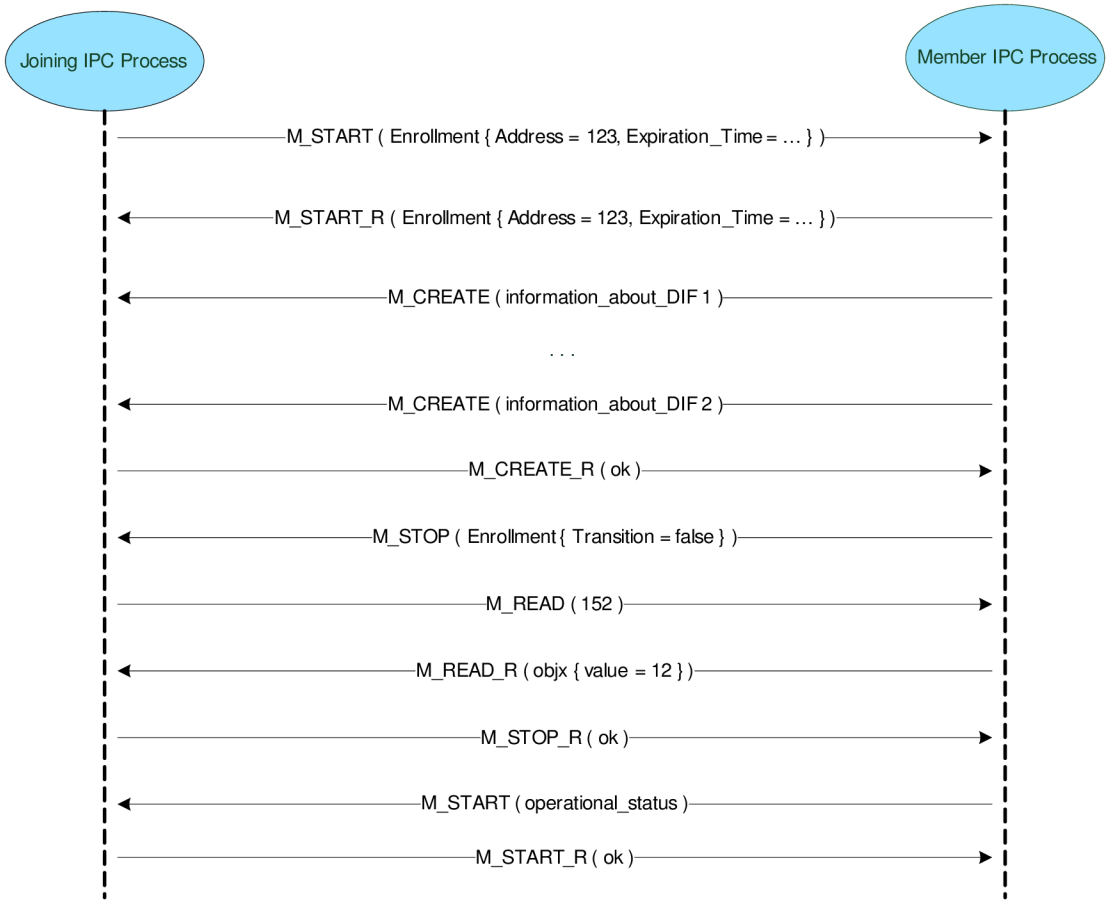


Figure 5.2: Enrollment communication when IPC Process was also member of a DIF

with address, which was assigned to it before. The Member sends `M_START_R Enrollment` response with the address and confirmation that it is all right. This is valid in case that the initiator has still valid address (its expiration time still not expired) or if the initiator is trying to enroll another member for support multipath connection.

For case when the member is rejoining to the DIF with address that expired, the member will send new one to it. The difference is shown in picture 5.3. The rest of communication is similar to the case above.

The `M_START Enrollment` request may contain list of known informations such as objects created when it enroll for the first time. Responding process may check that information if it is still valid and may shorten the part when objects are created or entirely skip it. Also the initiator do not need to read so much information and this also shorten whole

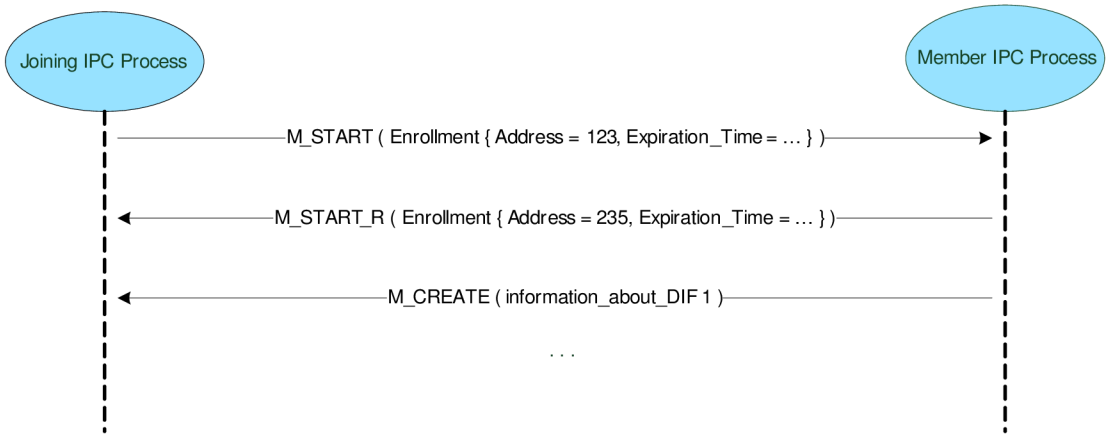


Figure 5.3: Enrollment communication when IPC Process was also member of a DIF

Enrollment. At best case these two parts could be entirely skipped and the Enrollment should be as quick as possible.

The following diagrams is designed based on known informations described in this chapter. They covers all of the possible cases.

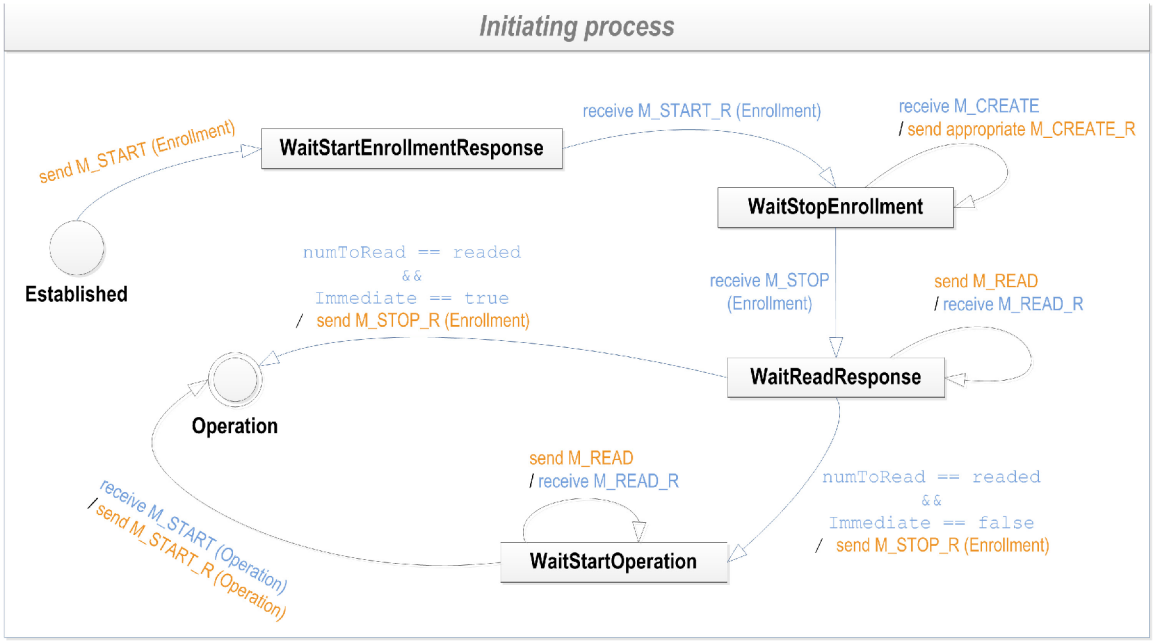


Figure 5.4: Initiating process Enrollment State Diagram

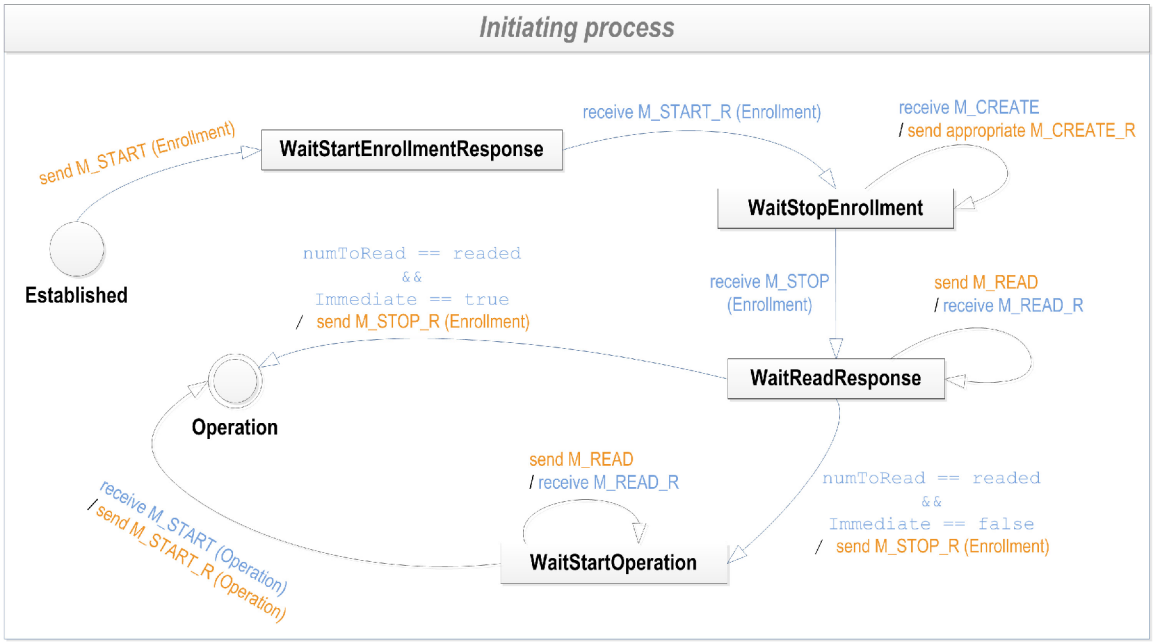


Figure 5.5: Responding process Enrollment State Diagram

Chapter 6

OMNeT++

OMNet++ [1] is a discrete event simulation environment which is primary used for a simulation of networks. The network simulation is not the only field of this environment, its also used for the simulation of IT systems as well as hardware architectures.

When modeling in the OMNeT++, the main concept is in modules that are hierarchicaly pludged and communicating with each other. The programming of logic of this modules is used C++ language and for the defining structure of modules is used special language called NED.

The modules are communicating through messages that may come from a module to another module or could be sent from a module to a module itself. The messages are sent through the specially defined input and output gates. Between that gates connections are created that all communication among modules go through.

The OMNeT++ is running on every Unix-like platforms as well as on Windows, Linux and Mac OS X.

Chapter 7

Description of Implementation

This chapter contains description of the implementation of Enrollment module and its inclusion within IPC process.

7.1 Module design

The Enrollment take place within the creating management connection between two IPC processes. There is no doubt, that Enrollment module belongs to the IPC process. It also needs to communicate with other modules within one IPC process such as the RIB Daemon that is a primary user of the CDAP within the IPC process [6].

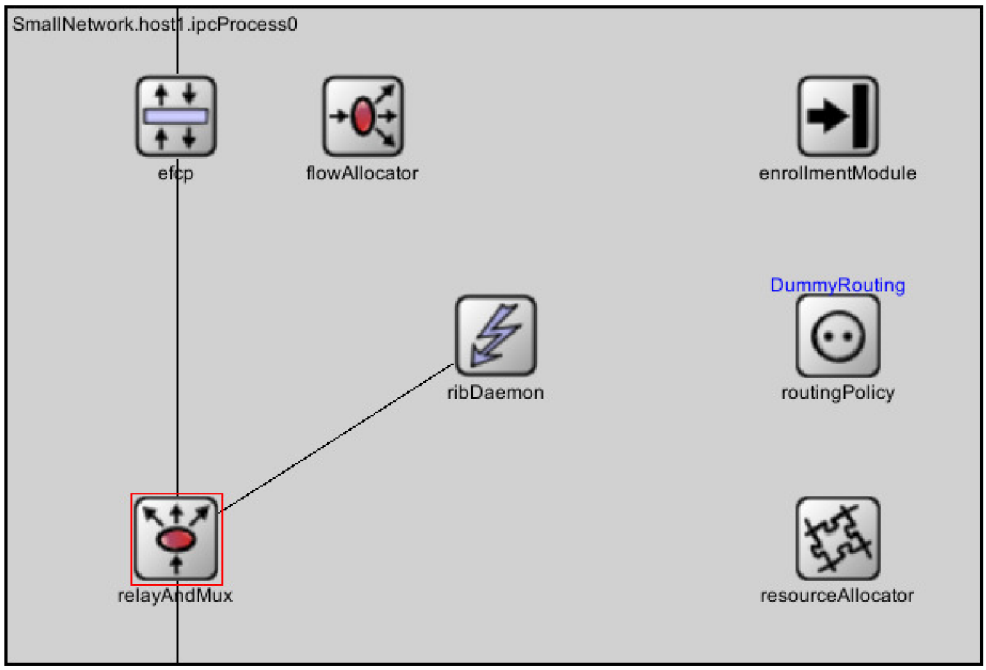


Figure 7.1: IPC process in OMNeT++

Because to one IPC process, which is also member of DIF, could be connecting more than one IPC processes, there is a need to keep information about the state and other information

associated with each management flow. For this purpose is created `enrollmentStateTable` which is the component of this module as it is shown in Figure 7.2.

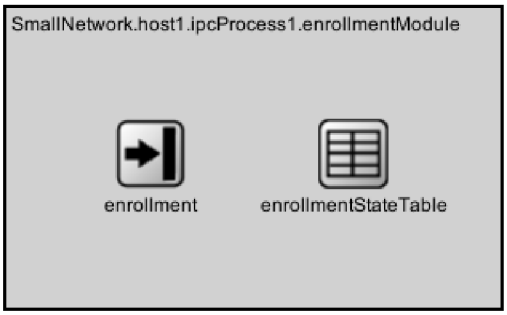


Figure 7.2: Enrollment module from inside in OMNeT++

The `enrollment` included in the `enrollmentModule` contains the part of the pattern machine that reacts on received messages. This module is currently processing all the communication associated with the CACE and the Enrollment. It is not necessary to have a separate module for CACE because the Enrollment starts right after the connection is established over the management flow.

7.2 Implementation

All the implementation is managed by object oriented paradigm. Therefore, every important part is represented by its own class. This include CDAP messages, table entries and also listeners associated with signals.

The Enrollment module is implemented as a pattern state machine for both sides that reacts on received signals that in most cases carry the CDAP messages as an input. It maintain information about the authentication (`authType`, `authName`, `authPassword`, `authOther`). This information as well as maximum connection retries (`maxConRetries`) is configurable and it is taken from parameters included in the `enrollment.ned` file.

This module also includes a function to validate authentication information held by `M_CONNECT` message. Two types of the authentication are supported namely: none and simple password.

Each function is getting and setting all information from/to the `EnrollmentStateTable` which contains `std::list<>` of `EnrollmentStateTableEntry`. This entry carry all necessary information about states (`conStatus` for CACE and `enrollStatus` for Enrollment) and other information such as current connection retries (`conRetries`), if Enrollment stops immediate (`immediateEnrollment`). This information is associated with given management flow.

Receiving and sending messages is provided by the RIB Daemon. Received and sended messages are forwarded between modules by signals that one modul emits and appropriate listener of this signal received and forward to its module.

The whole implementation follows the description provided in chapters 4 and 5.

Currently the CACE and the Enrollment happens over the standard data transfer flows. The common application connection phase starts after the Flow allocator instance (FAI) emits the signal that presents the successful allocation of the flow. This is caused by that

the RINA simulator is still being developed. However the CACEP and the Enrollment is working even over that flow and it could be demonstrated.

7.3 Objects

According to the Enrollment were created two special classes namely `EnrollmentObj` and `OperationObj`. The `EnrollmentObj` is used for start and stop Enrollment and it is currently carrying information about address (`address`), the current address (`currentAddress`), address expiration time (`addressExpirationTime`) and if the Enrollment stops immediate (`immediate`). Another object is the `OperationObj` that is used to start operational state.

Chapter 8

Model Validation

This chapter contains a basic validation according to avaliable resources that are the only specifications of the CACEP [12] and the Enrollment [5].

The behavior of implemented model is deterministic. It was tested on two different simulation examples repeatedly with the same result. The main factors to control was if each side react on a received message by the transition to the appropriate state and if it answers with the expected message.

The CACEP starts right after the flow is successfuly allocated. The initator sends the `M_CONNECT` with all the defined values assigned. The member validates the authentication values. If none authentication is used, it sends the positive connect response always. If the authentication by password is used, the member appropriate response. If it is negative, the initiator then sends connect retries up to max connect retries.

The Enrollment proceeds in the way it was designed in chapter 5.3.2. It is expected that there is no need to create or read any additional information. However all of the states for both the initiator and the member progress in the right order. The Enrollment is in this case as quick as possible and comply the design.

The successfully authenticated connection is enrolled and it finishes with the state `ENROLLED`. Both phases strictly follow the design. According to that and according to specifications that corresponds to the draft, the model could be marked as valid.

Chapter 9

Conclusion

The aim of this thesis was to analyse and to describe the connection of IPC process to a layer (DIF). It was found that this consecution includes two phases of the communication: the Common Application Connection phase and the Enrollment phase that are described in more detail here. The communication in these two phases is provided by Common Distributed Application Protocol which is described here as well.

In this work the design was created out of these two phases according to available information. The Enrollment was created that controls the communication within both phases. The resulting modul is the part of the RINA Simulator developed in OMNeT++ on our faculty within Pristine project.

The Enrollment phase currently supports only so called pragmatic case. In that case the requestor was member of a DIF and it tries to rejoin. Currently, other cases are not possible because the whole simulator is still being developed. However, the implemented part of the model is working and validated.

Further developement of the Enrollment module will deal with the other case of the Enrollment and to support a proper deallocation. In addition, the CACEP occures not only within DIF but as well within DAF. There is also a need to have a module to support the CACE phase or it could be a part of a module.

Bibliography

[1] Omnet++ community site. http://www.omnetpp.org. Accessed: 2015-1-9.

[2] The pouzin society - building a better network. http://pouzinsociety.org.

[3] Steve Bunch. Cdap - common distributed application protocol reference, December 2010. unpublished.

[4] John Day. *Patterns in Network Architecture: A Return to Fundamentals*. Prentice Hall, 2008. ISBN-13: 978-0-132-25242-3.

[5] John Day. Patterns in network architecture - recursive ipc network architecture - basic enrollment specification, 2012.

[6] John Day. Recursive ipc network architecture - the interina reference model - part 3: Distributed interprocess communication - chapter 1: Fundamental structure, 2012.

[7] John Day. Recursive ipc network architecture - the interina reference model - part 3: Distributed interprocess communication - chapter 2: Dif operations, 2012.

[8] John Day. Patterns in network architecture - recursive ipc network architecture - the interina reference model - part 1: Basic concepts of distributed systems, 2013.

[9] John Day. Recursive ipc network architecture - the interina reference model - part 2: Distributed applications - chapter 1: Basic concepts of distributed applications, 2013.

[10] Eleni Trouva John Day. Recursive ipc network architecture - the interina reference model - part 2: Distributed applications - chapter 2: Introduction to distributed management systems, 2014.

[11] Karim Mattar John Day, Ibrahim Matta. Rearch'08 - re-architecting the internet (madrid, spain, december 2008). 2008.

[12] Eleni Trouva Steve Bunch, John Day. Patterns in network architecture - recursive ipc network architecture - common application connection establishment phase (cacep), 2012.

Appendix A

CD index

`RINA/src/`	Source codes of all current RINASim modules (including enrollmentModul).
`omnetpp-4.5-src.tgz`	OMNeT++ simulator installation package for Linux.
`tex/`	LaTeX source code of this thesis.
`project.pdf`	Electronical version of this thesis in PDF format.
`README`	CD content, installation guide.

Table A.1: CD index