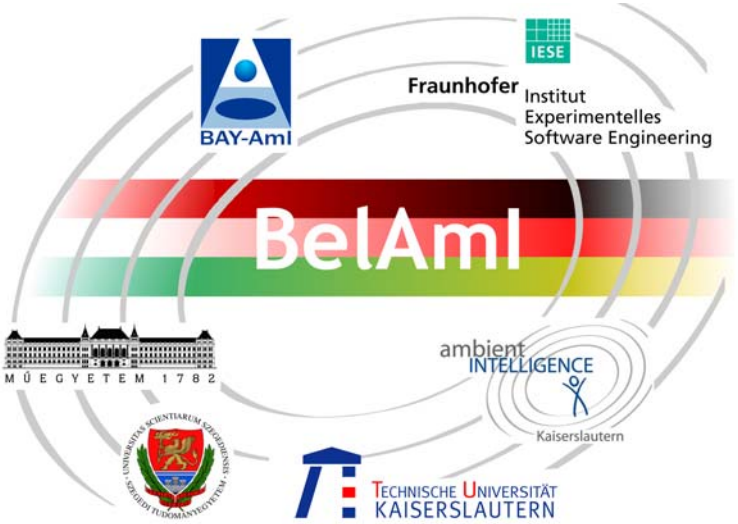
Fraunhofer Institut Experimentelles Software Engineering

Technische Universität Kaiserslautern

AmICom - Formally specified service platform for ambient intelligence networks

Authors:
Ingmar Fliege
Networked Systems Group
University of Kaiserslautern

Jan Koch
Robotics Research Group
University of Kaiserslautern



Deliverable
D2.6.1

Date: 12 June 2007
Version: 1.0
Status: Final
Classification: Internal

A publication by Fraunhofer IESE

Fraunhofer IESE is an institute of the Fraunhofer Gesellschaft.
The institute transfers innovative software development techniques, methods and tools into industrial practice, assists companies in building software competencies customized to their needs, and helps them to establish a competitive market position.

Fraunhofer IESE is directed by
Prof. Dr. Dieter Rombach (Executive Director)
Prof. Dr. Peter Liggesmeyer (Director)
Sauerwiesen 6
67661 Kaiserslautern

Abstract

Intelligent environments are based on services that are deployed on hardware nodes that communicate among each other. Generic and light-weight communication middleware for service interaction is the basis of the ubiquitous idea. In this report, AmICom, a new light-weight communication middleware for dynamic ambient applications is presented. This middleware is already in use on several nodes within a prototypical assisted living environment.

Keywords: Ambient intelligence, network, communication, ad-hoc, distributed services

Table of Contents

1 Introduction

Ambient intelligence environments consist of different type of nodes, which communicate through wired networks or by wireless mobile ad-hoc networks that come into existence by the mere presence of nodes that form a self-organizing network. They support distributed user applications in various areas of ambient intelligence, including professional work, leisure activities, public health, and transportation. For these user applications, it is essential to provide their distributed services independently from the current network architecture and without reconfiguration of the network.

This report presents *AmICom*, the tailored lightweight communication middleware for ambient intelligence networks, which is based on distributed services. Each application in the network may register services, which are then available to all nodes in the network. Thereby, all other applications can subscribe to this service to form a multicast communication group.

Applications interchange messages by use of the *AmICom*, which allows an abstraction from the distributed system, by delivering valid messages to applications on the local node or to application on other nodes in the network. A message is valid, if the destination address of the massage corresponds to the name the application has registered.

2 Requirements

2.1 Basic Functionality

- AmICom instance:
  AmICom is instantiated once on each node and will be provided as an executable that run on linux and Windows platforms. Multiple applications can use the communication service of the AmICom.

- Service for applications:
  A class for Java and Windows is provided, which allows the interaction with the running AmICom instance on the local node.

- Application registration by name:
  Each application registers itself at the local AmICom, by providing a unique name.
  bool AmICom::Register(String name)
  AmICom keeps a register of applications present on the node. This registration is a requirement for further communication with the AmICom. The function returns false if the connection to the local AmICom instance failed.

- Application subscribe by name:
  Each application may subscribe a service in the network by use of its unique name.
  bool AmICom::Subscribe(String name)
  The subscription triggers a communication between this AmiCom instance and any other instance in the network, which provides a service with this name. Additionally to the dissolving of name clashes, a connection between both AmiCom nodes is established and observed. *The function returns false if the connection to the local AmICom instance failed or a service with this name is not available.*

- Sending unicast messages:
  Sends a message to another local or remote application:
  bool AmICom::Send(String destination-name, byte[] data, int size)
  The provided destination-name must corresponds to the name, an application has registered, in order to receive that message.

- Receiving messages:
  Messages transmitted over the network are queued in the AmICom, if an application with a corresponding name has been registered before. The registered application can gather a message, in order of reception from this queue.
  struct message_data AmICom::GetData()
  - returns null if buffer is empty

2.2 Future functionality

Convenience methods for simple data types

To access ints, floats, chars and strings in data packets, we will provide some convenience methods for data transformation. Here, the description of data types using ASN.1 and the Encoding/Decoding using BER, DER or PER is proposed.

Multi-NIC support (work in progress)

Enable the multi-hop communication in ad-hoc networks will be provided by support of multiple hardware interfaces and appropriate routing protocols

2.3 Optional functionality (extensions)

Possible further extensions are listed below. The realisation of further extensions of the AMICOM depends on complexity, effort and available resources. Each extension requires further requirement analysis and agreements.

- Service availability check (work in progress)
  Applications may register a periodically check of the availability of a particular application at their AmICoM. Monitored applications are requested to answer to certain refresh messages. If this fails or any other loss of interaction in the distributed system is detected, other applications may be notified.

- Check for name clashes (work in progress)
  As communication is done via names, these have to be unique. To avoid application name clashes at runtime (user errors), AmICoM probes at registration time if the given name is already present and refuses registration in this case. This would exclude the possible extension "Group management"

- Group management
  Applications registering with equal names, form a group of communicating applications. A message with the destination-name of the

group will be delivered to all members of the group. This extension would exclude the possible extension "Check for name clashes"

- Observation of data messages
  The transmission of messages may be observed in order to check the successful delivery to one application in the network. Otherwise the transmission will continuously be repeated.

- Fragmentation and reassembly of data (work in progress)
  For a better support of large data packages from the application (e.g. video frames or large XML descriptions), the data must be segmented in multiple fragments. The size of the data must accomplish the requirements of the used communication technology.

3 AmICom Middleware

The AmICom middleware is specified using the formal specification language SDL-96. The transpiler ConTraST automatically generates an implementation in C++. Together with the corresponding runtime environment and a set of generic communication modules an executable for multiple platforms can be generated.

3.1 Architecture

Figure 1 shows the general architecture of each node in the ambient environment. Depending on the available hardware resources, there are possibly several applications, which may register or subscribe to a particular service in the network.

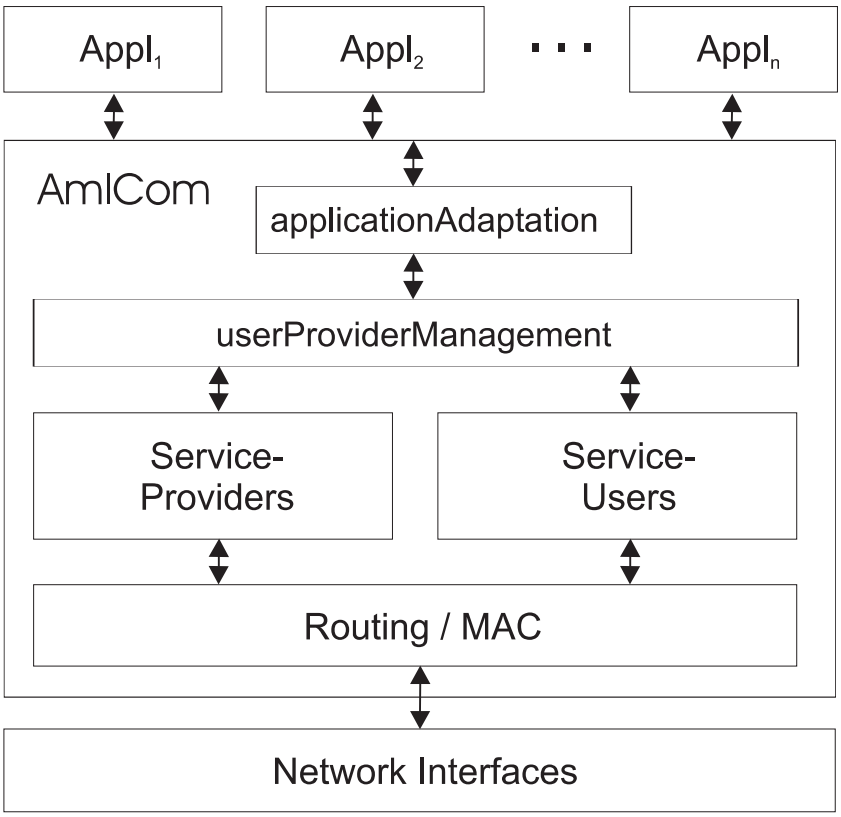


Figure 1: AmICom architecture

For each request with a new service name, a *ServiceProvider* or a *ServiceUser* is created, representing a communication endpoint for the application. The communication between a ServiceProvider and all of its subscribed ServiceUsers is observed in order to detect the loss of communication or the failure of a node in the network. In case that the failure cannot be recovered, the subscribed applications are notified.

3.2 Services

One objective of AmICom is to keep the programming interface as simple as possible to facilitate the familiarization with the interface, increase the acceptance and allow the portage to multiple hardware platforms. The API to the AmICom middleware is realised by C functions or as methods of a class in all object-oriented languages:

- bool: REGISTER(string: name)
- bool: UNREGISTER(string: name)
- bool: SUBSCRIBE(string: name)
- bool: UNSUBSCRIBE(string: name)
- bool: SEND(string: name, bytes: data)
- bool: RECEIVE(string: name, bytes: data)

Figure 2 shows a typical scenario for the use of the AmICom API. The REGISTER call (respectively UNREGISTER) allows an application to register a service in the AmI network with a given name. Each application interested in this particular service may subscribe to this service with a SUBSCRIBE(name). AmICom observes these operations and gives a feedback, whether the request could be fully performed. The REGISTER may fail if a service with this name is already been registered in the network. The SUBSCRIBE fails, when the requested service is currently not available.

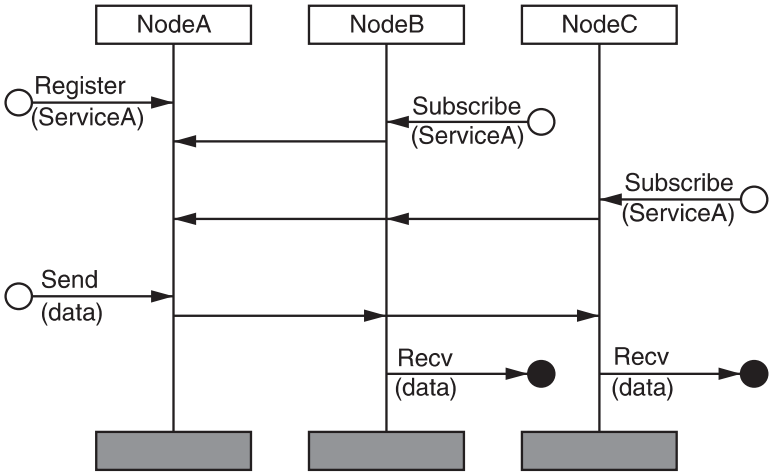


Figure 2: Communication scenario

All applications that have registered or subscribed may SEND messages that are transmitted to all other associated application with this service. These messages can be consumed by a RECEIVE call. Additionally, the registration of callback functions is possible, allowing an immediate reception of messages and the asynchronous notification of service failure.

3.3 Platforms & Languages

The AmICom currently support the following platforms:

- Windows (Windows 2000 and newer)
- Linux (Kernel 2.4 and newer)

The AmICom API used by the application developer is available for:

- C
- C++
- Java

4 Behaviour and service description

Name: AmICom

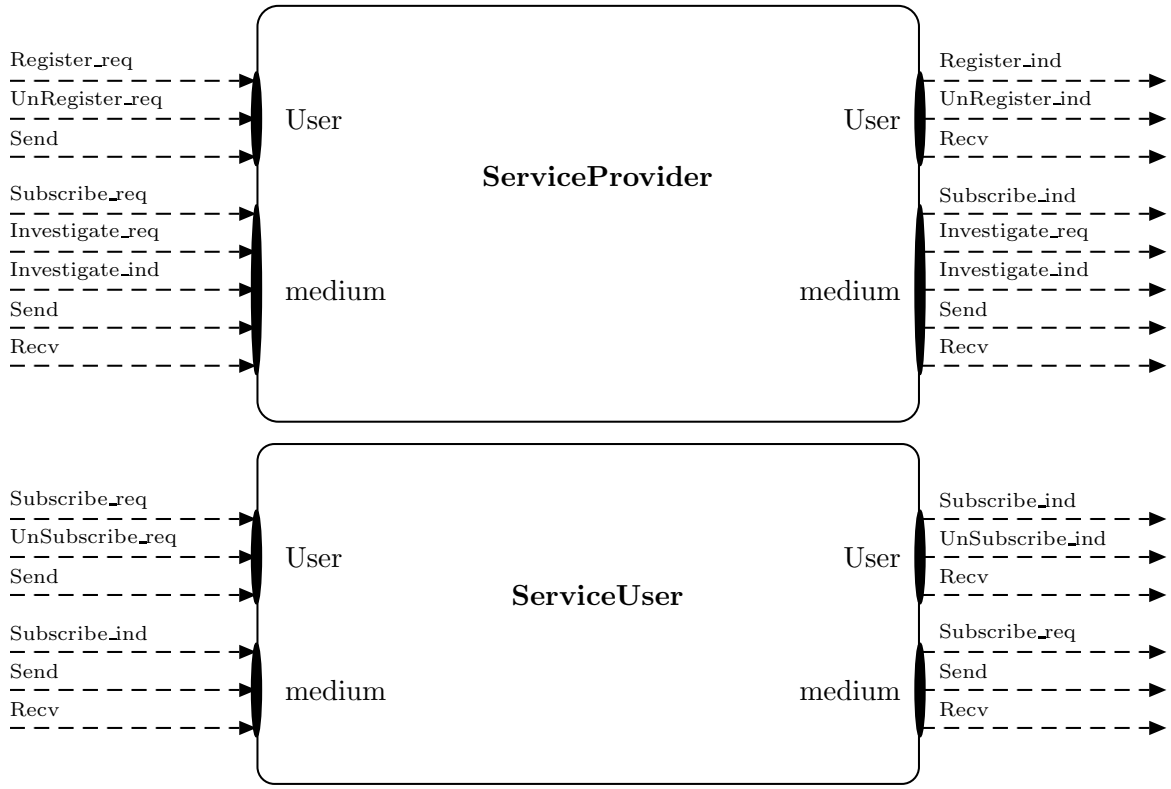
Version: 1.0

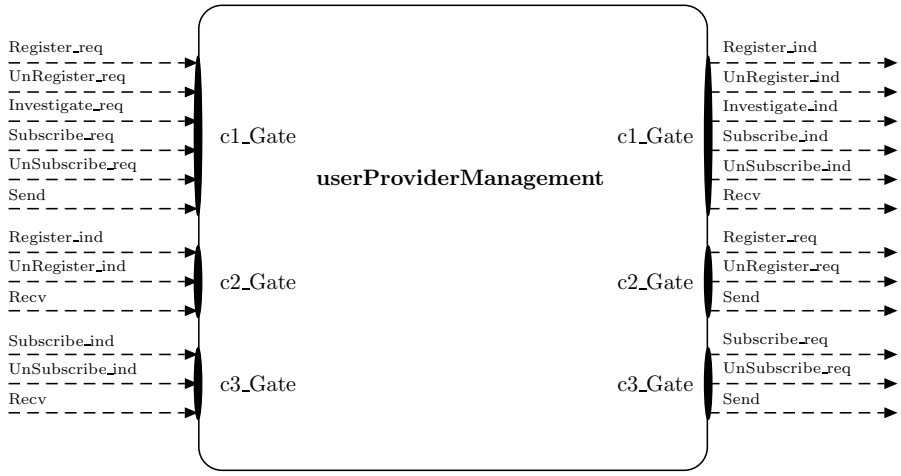
Author: Ingmar Fliege

Intent

These micro protocols specify the distibuted asymetric service platform: AmICom

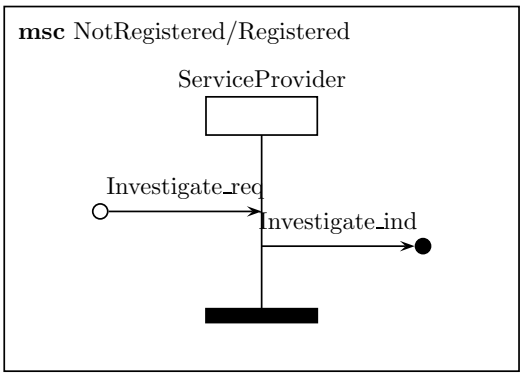
Interface signature




Register_req
UnRegister_req
Investigate_req
Subscribe_req
UnSubscribe_req
Send
Register_ind
UnRegister_ind
Recv
Subscribe_ind
UnSubscribe_ind
Recv
c1_Gate
c2_Gate
c3_Gate
userProviderManagement
c1_Gate
c2_Gate
c3_Gate
Register_ind
UnRegister_ind
Investigate_ind
Subscribe_ind
UnSubscribe_ind
Recv
Register_req
UnRegister_req
Send
Subscribe_req
UnSubscribe_req
Send

4.1 Interface behaviour

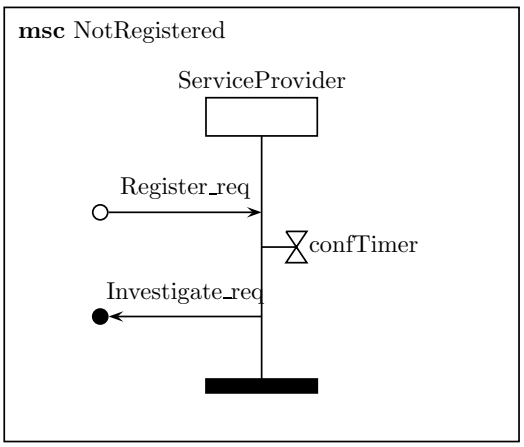
Scenario 1



Description:

The signal *Investigate_req* indicates a query for a service with the given name in the network. The signal *Investigate_ind* is the reply for the query. Since the name corresponds to one that was searched, the Investigate_ind is sent.

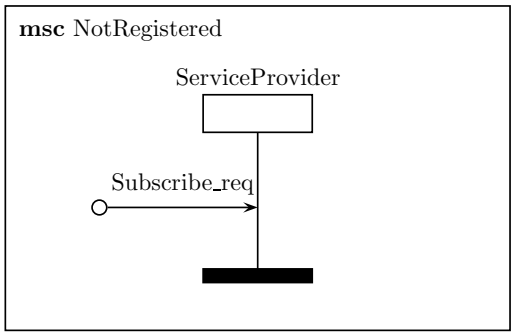
Scenario 2



Description:

The signal *Register_req* initiates the registration of a new service in the network. The timer *confTimer* is set for a duration of CONFTIMEOUT time units. Next, the signal *Investigate_req* searches for a service in the network, which has already registered a service with this name.

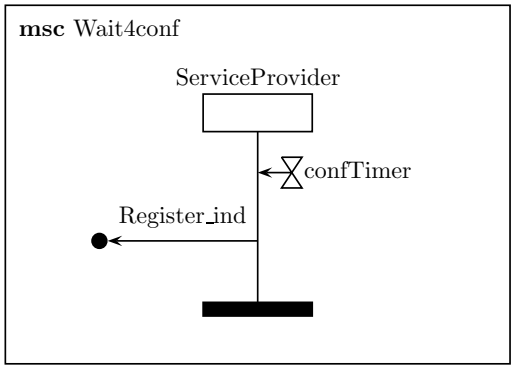
Scenario 3



Description:

The signal *Subscribe_req* initiates a subsription to a service with the given name. No further operations are performed.

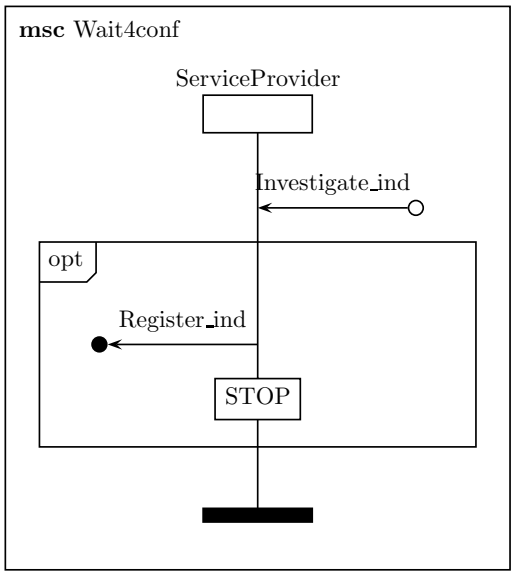
Scenario 4



Description:

The timer *confTimer* signals, that there is no other service in the network registered by this service name. The signal *Register_ind* indicates the successful registration of a new service in the network.

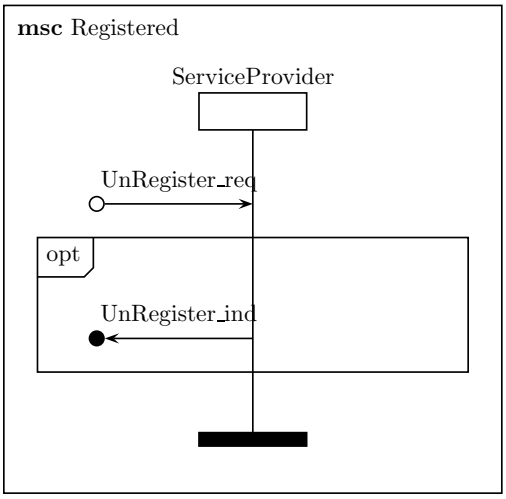
Scenario 5



Description:

The signal *Investigate_ind* is consumed and triggers the following actions: If this service name is already registered by another instance of this protocol, the signal *Register_ind* is sent and the process stops.

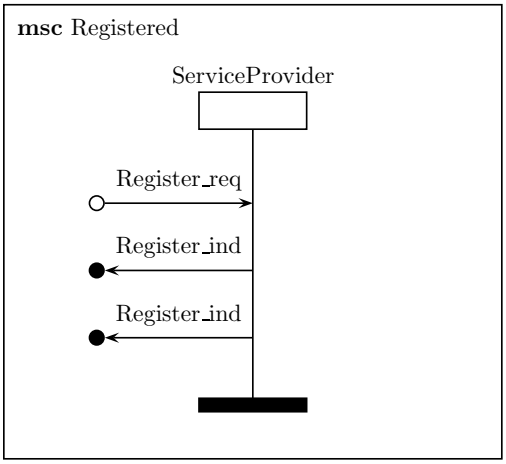
Scenario 6



Description:

A service with a well defined name is registered by the service provider. The signal *UnRegister_req* is consumed and triggers the following actions: If the application is known by this service, the signal *UnRegister_ind* indicates the cancellation of the registered service.

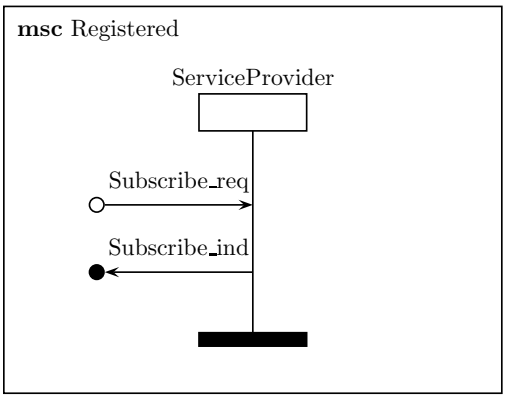
Scenario 7



Description:

A service with a well defined name is registered by the service provider. The signal *Register_req* initiates reregistration of this service (same name) by another application on the current node. The signal *Register_ind* indicates the success of registration to the new application, which has created the request. The signal *Register_ind* indicates the loss of registration to the first application.

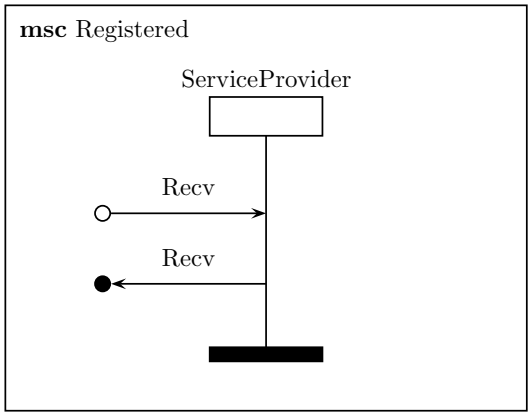
Scenario 8



Description:

A service with a well defined name is registered by the service provider. The signal *Subscribe_req* initiates a subscription to a service with the given name. The signal *Subscribe_ind* confirms that the subscription was successful.

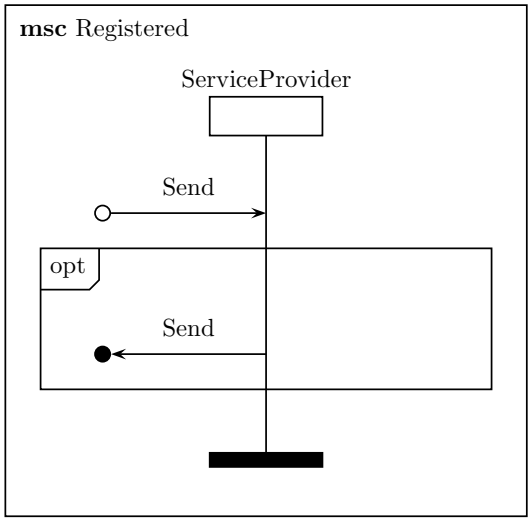
Scenario 9



Description:

A service with a well defined name is registered by the service provider. The signal *Recv* contains data for the application that was send by some service user in the network. The signal *Recv* indicates the reception of new data for the application.

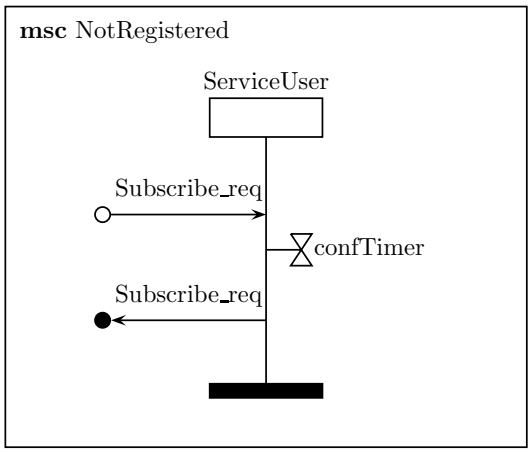
Scenario 10



Description:

A service with a well defined name is registered by the service provider. The signal *Send* contains data from the application that has to be sent to other applications in the network. If the service name corresponds to the name of this service, the signal *Send* transmits the data to all participating services in the network.

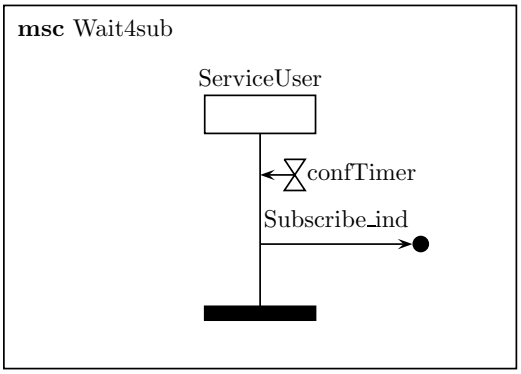
Scenario 1



Description:

No application has subscribed a service in the network. The signal *Subscribe_req* intiates the subscription at a service by name in the network. The timer *confTimer* is set for a duration of CONFTIMEOUT time units. Next, the signal *Subscribe_req* is sent to all service providers in the network in order to subscribe the specified serivce.

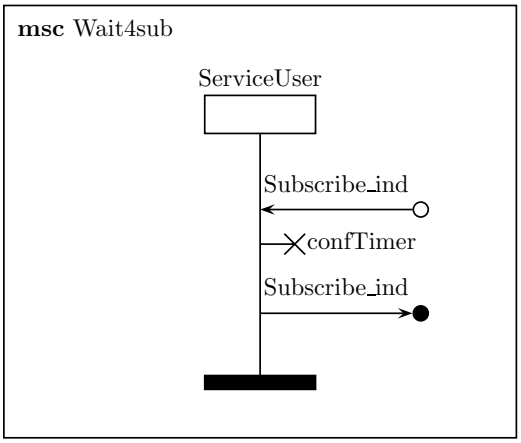
Scenario 2



Description:

The timer *confTimer* indicates that the requested service was not found. The signal *Subscribe_ind* indicates the failure of subscription.

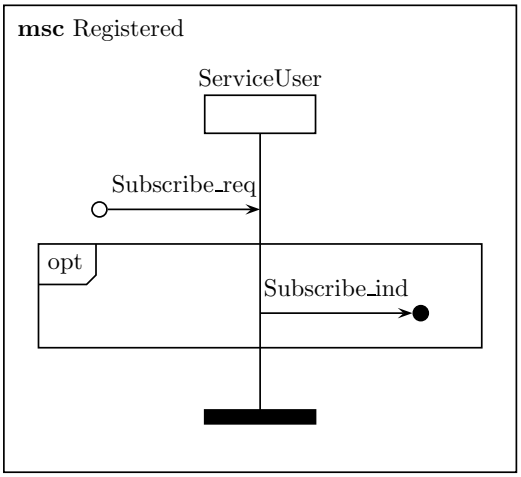
Scenario 3



Description:

The signal *Subscribe_ind* is the confimation of subscription at the requested service. The timer *confTimer* is reset. Next, the signal *Subscribe_ind* indicates the success of subscription.

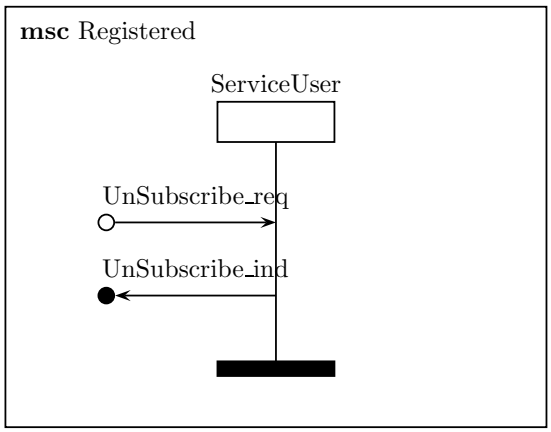
Scenario 4



Description:

A service user has subscribed a service in the network. The signal *Subscribe_req* indicates that another application whishes to subscribe a service which has already been subscribed by another application. If the requested service name corresponds to the service name this instance has subscribed, the signal *Subscribe_ind* confirms the successful registration.

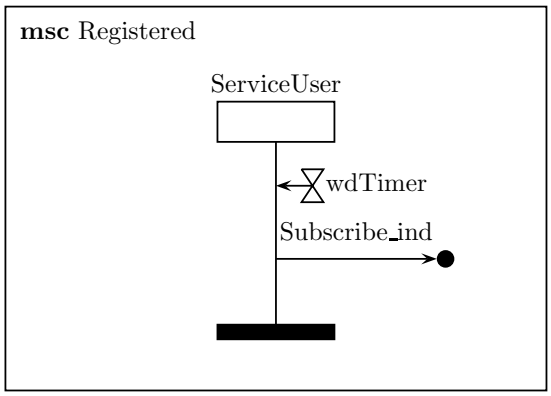
Scenario 5



Description:

A service user has subscribed a service in the network. The signal *UnSubscribe_req* initiates the cancelation of a service. The signal *UnSubscribe_ind* is the confirmation of the unsubscribe operation.

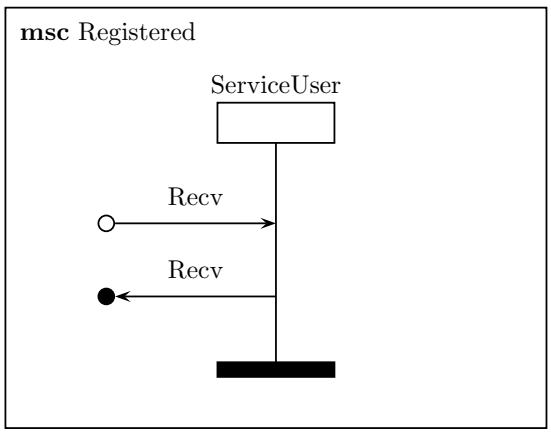
Scenario 6



Description:

A service user has subscribed a service in the network. The timer *wdTimer* is consumed and the signal *Subscribe_ind* is sent.

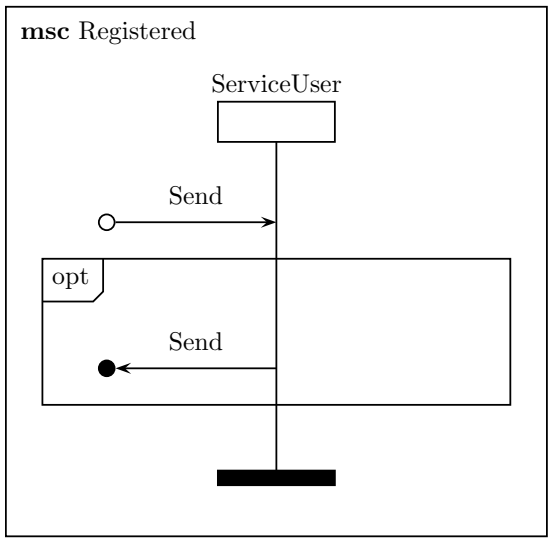
Scenario 7



Description:

A service user has subscribed a service in the network. The signal *Recv* contains data for the application that was send by some node in the network. The signal *Recv* is sent.

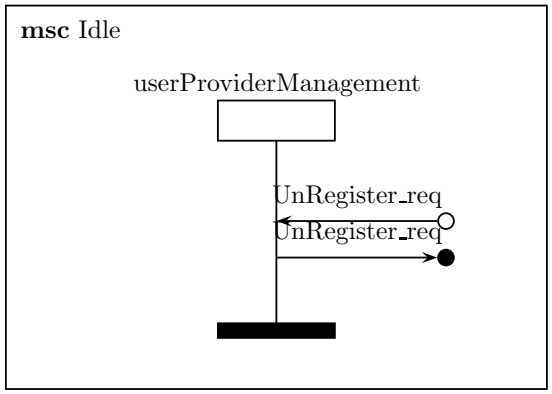
Scenario 8



Description:

A service user has subscribed a service in the network. The signal *Send* contains data from the application that has to be sent to other applications in the network. If the application has registed at the specified service, the signal *Send* transmits the data to all participating services in the network.

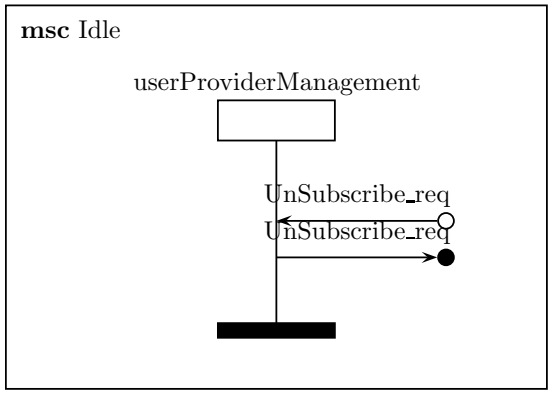
Scenario 1



Description:

The signal *UnRegister_req* initiates the cancellation of a service registration by the application. Otherwise, the signal *UnRegister_req* is sent.

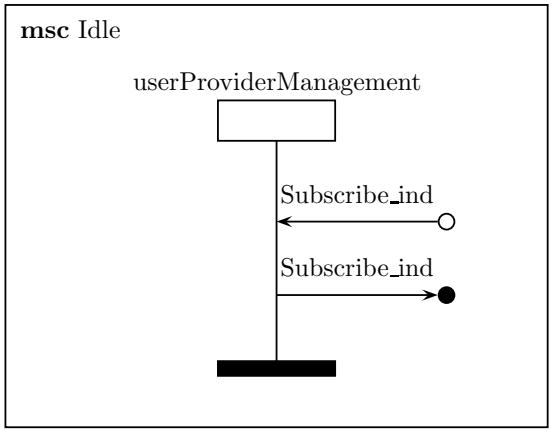
Scenario 2



Description:

The signal *UnSubscribe_req* initiates the cancellation of a service subscription by the application. Otherwise, the signal *UnSubscribe_req* is sent.

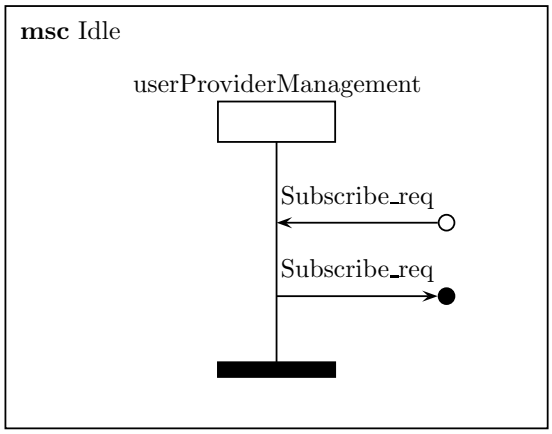
Scenario 3



Description:

The signal *Subscribe_ind* indicate, that the subscription at the service has completed. The signal *Subscribe_ind* is the answer to the application with a return value indicating the success or failure.

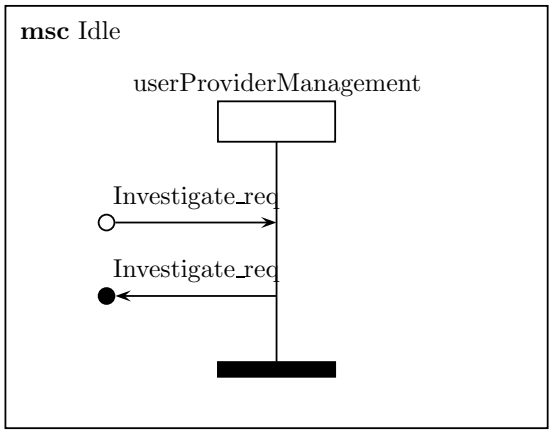
Scenario 4



Description:

The signal *Subscribe_req* initiates the subscription at a service with the given name. The signal *Subscribe_req* instructs the local service user to be responsible for the subscription at a service in the network.

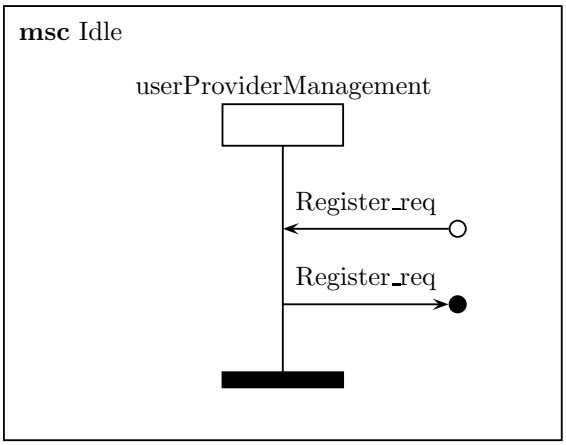
Scenario 5



Description:

The signal *Investigate_req* searches a serice by name in the network. The signal *Investigate_req* is sent.

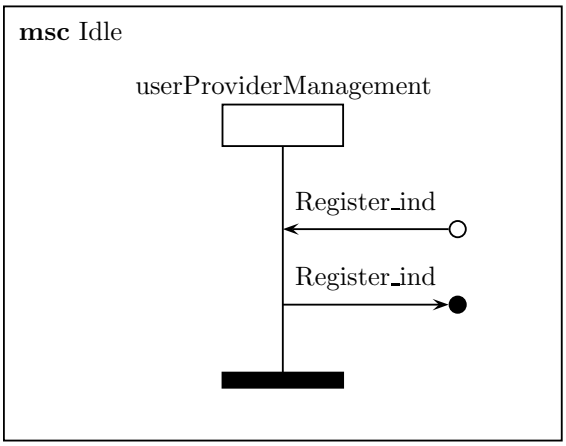
Scenario 6



Description:

The signal *Register_req* initiates the registration of a new service with the given name. The signal *Register_req* instructs the local service provider to be responsible for the service with the given name.

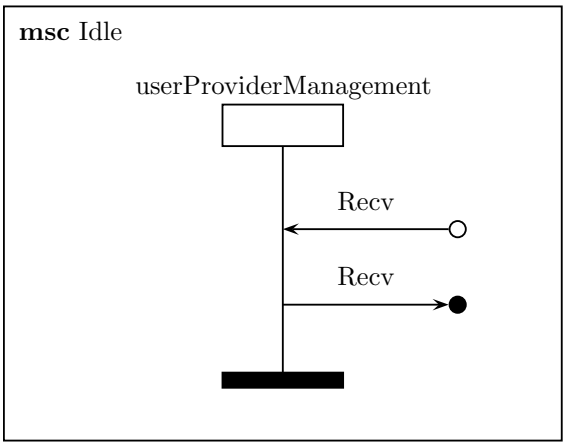
Scenario 7



Description:

The signal *Register_ind* indicate, that a new registration of service has completed. The signal *Register_ind* is the answer to the application with a return value indicating the success or failure.

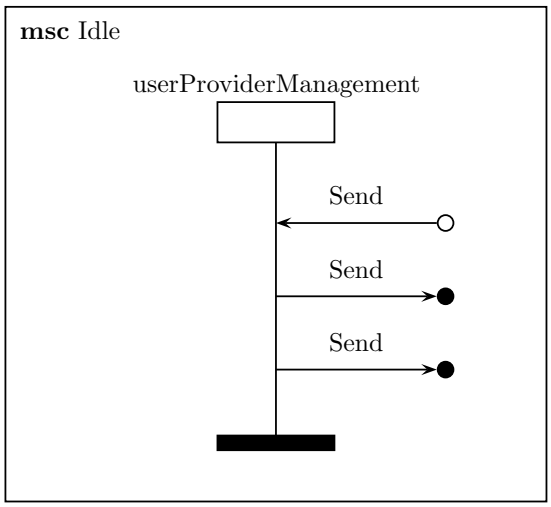
Scenario 8



Description:

The signal *Recv* contains data from some application in the network. The signal *Recv* forward the data to the appropriate application.

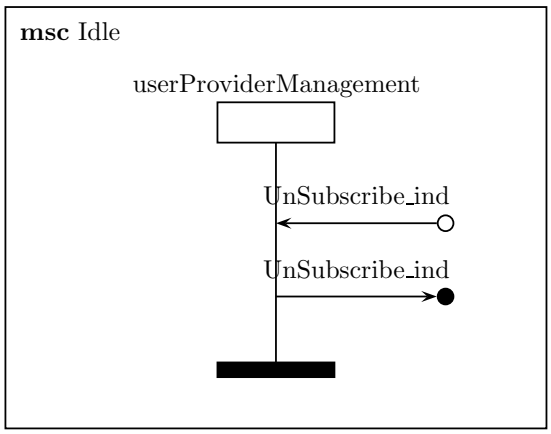
Scenario 9



Description:

The signal *Send* contains data from the application. The signal *Send* instructs the serice provider to send the data and the signal *Send* instructs the serice user to send the data.

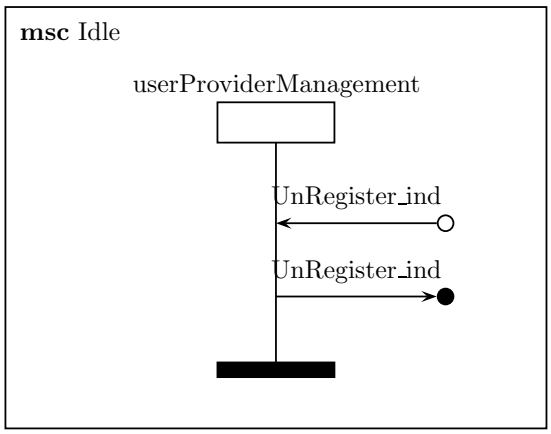
Scenario 10



Description:

The signal *UnSubscribe_ind* provides the result of the unsubscribe operation. The signal *UnSubscribe_ind* sends the result to the application.

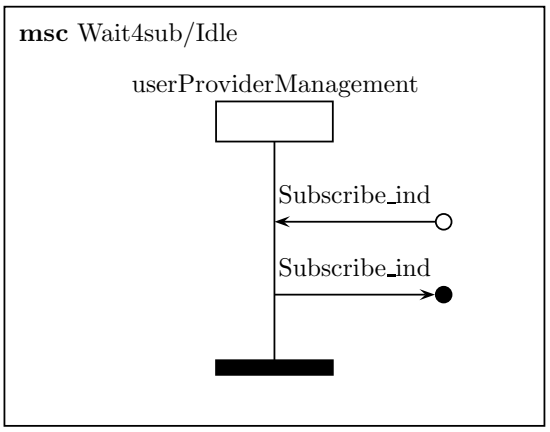
Scenario 11



Description:

The signal *UnRegister_ind* provides the result of the unregistration. The signal *UnRegister_ind* sends the result to the application.

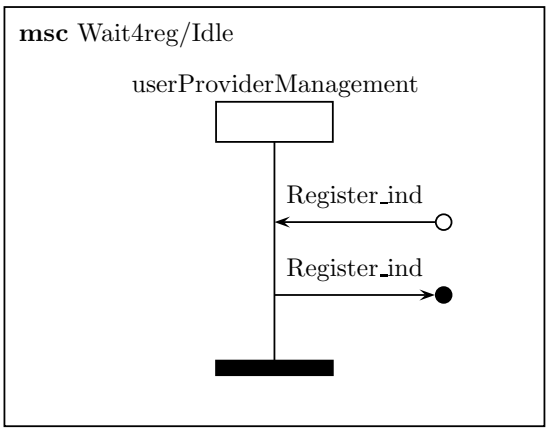
Scenario 12



Description:

The signal *Subscribe_ind* indicate, that the subscription at the service has completed. The signal *Subscribe_ind* is the answer to the application with a return value indicating the success or failure.

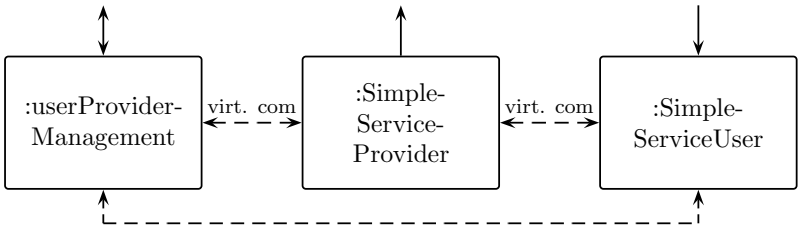
Scenario 13



Description:

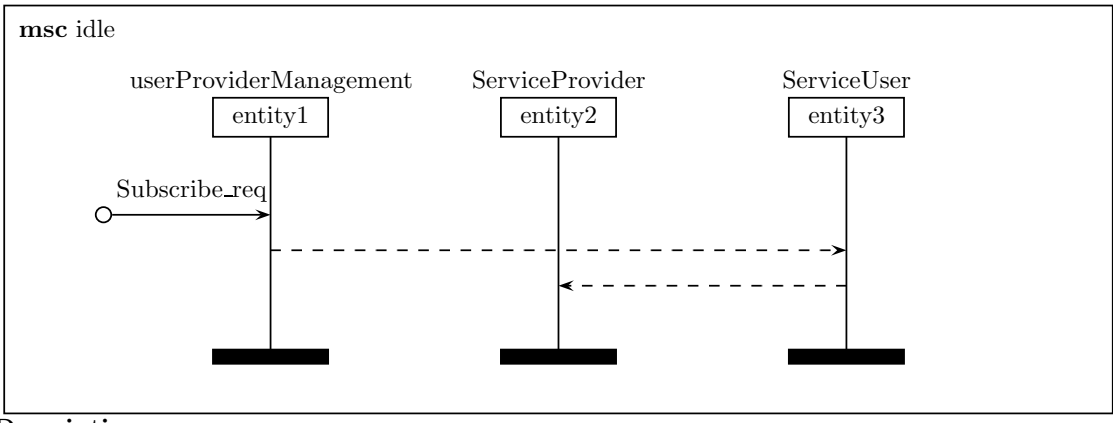
The signal *Register_ind* indicate, that a new registration of service has completed. The signal *Register_ind* is the answer to the application with a return value indicating the success or failure.

Architecture



4.2 Provided Service

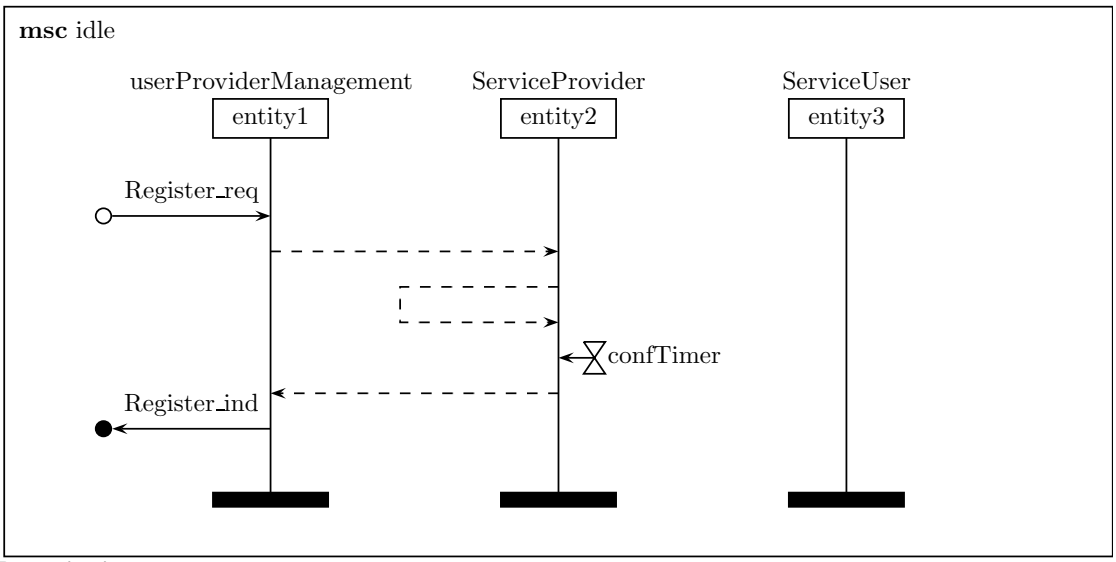
Scenario 1



Description:

No application has subscribed a service in the network.
The signal *Subscribe_req* initiates the subscription at a service with the given name. No further operations are performed.

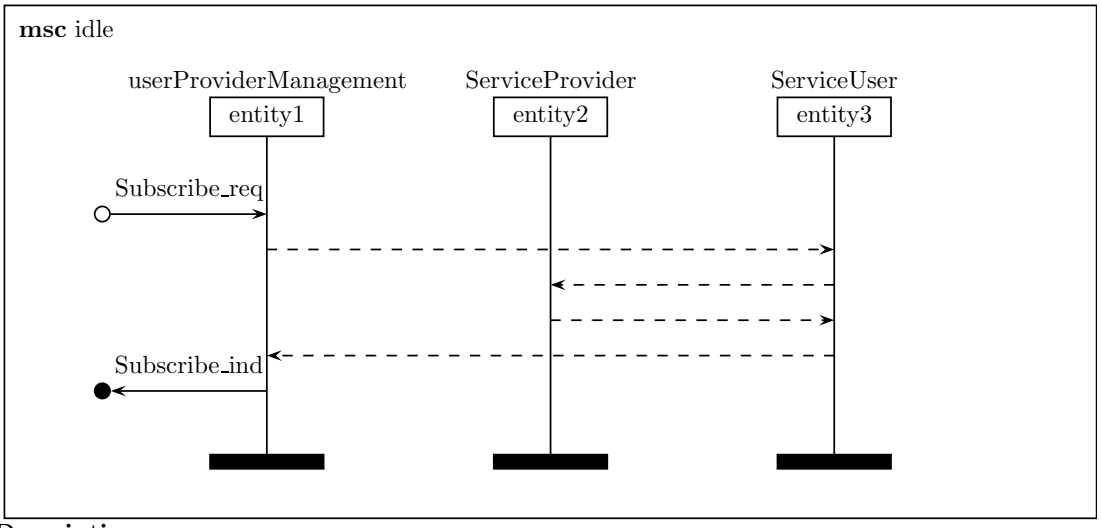
Scenario 2



Description:

No application has subscribed a service in the network.
The signal *Register_req* initiates the registration of a new service with the given name. The timer *confTimer* signals, that there is no other service in the network registered by this service name. The signal *Register_ind* is the answer to the application with a return value indicating the success or failure.

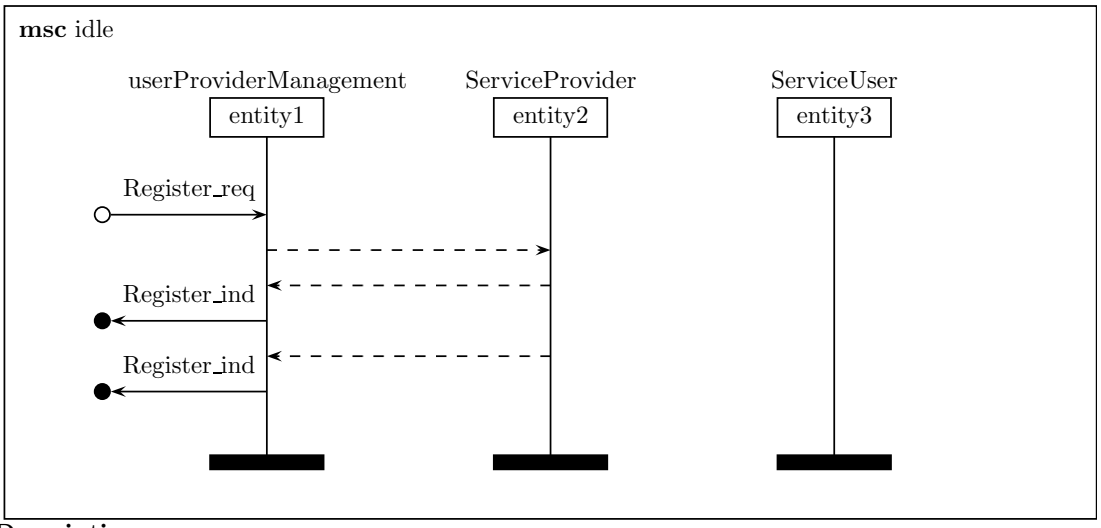
Scenario 3



Description:

A service with a well defined name is registered by the service provider. No application has subscribed a service in the network.
The signal *Subscribe_req* initiates the subscription at a service with the given name. The signal *Subscribe_ind* is the answer to the application with a return value indicating the success or failure.

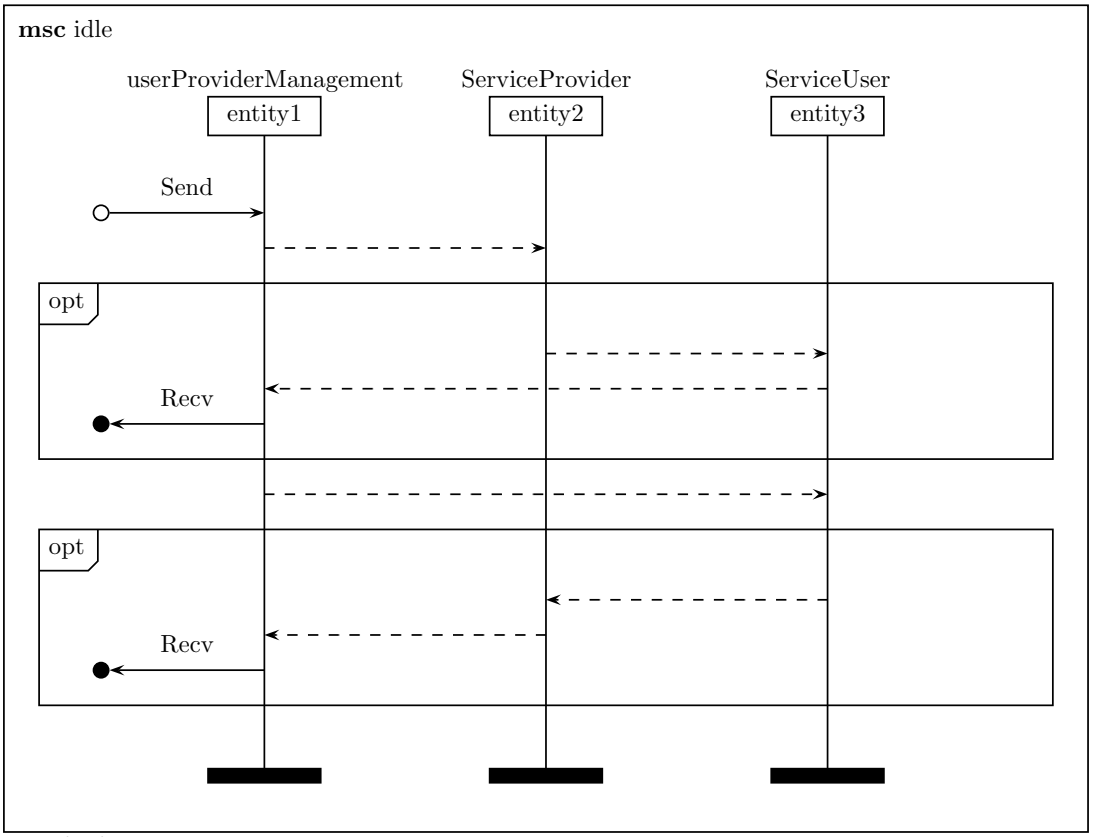
Scenario 4



Description:

A service with a well defined name is registered by the service provider. A service user has subscribed a service in the network.
The signal *Register_req* initiates the registration of a new service with the given name. The signal *Register_ind* is the answer to the application with a return value indicating the success or failure. The signal *Register_ind* is the answer to the application with a return value indicating the success or failure.

Scenario 5



Description:

A service with a well defined name is registered by the service provider. A service user has subscribed a service in the network.
The signal *Send* contains data from the application. If the service name corresponds to the name of this service, the signal *Recv* forward the data to the appropriate application. If the application has registed at the specified service, the signal *Recv* forward the data to the appropriate application.

Imported and exported definitions

Used packages

– *none* –

Required definitions

- Data type AddressType
- Signal Register_req(Integer,::AddressType::Newtype::AddressType)
- Signal UnRegister_req(Integer,::AddressType::Newtype::AddressType)
- Signal Send(Integer,::AddressType::Newtype::AddressType,Octet_string)

- Signal Recv(Integer,::AddressType::Newtype::AddressType,Octet_string)
- Signal Subscribe_req(Integer,::AddressType::Newtype::AddressType)
- Signal Investigate_req(Integer,::AddressType::Newtype::AddressType)
- Signal Investigate_ind(Integer,::AddressType::Newtype::AddressType,Boolean)
- Signal Subscribe_ind(Integer,::AddressType::Newtype::AddressType)
- Signal UnSubscribe_req(Integer,::AddressType::Newtype::AddressType)
- Signal UnSubscribe_ind(Integer,::AddressType::Newtype::AddressType)

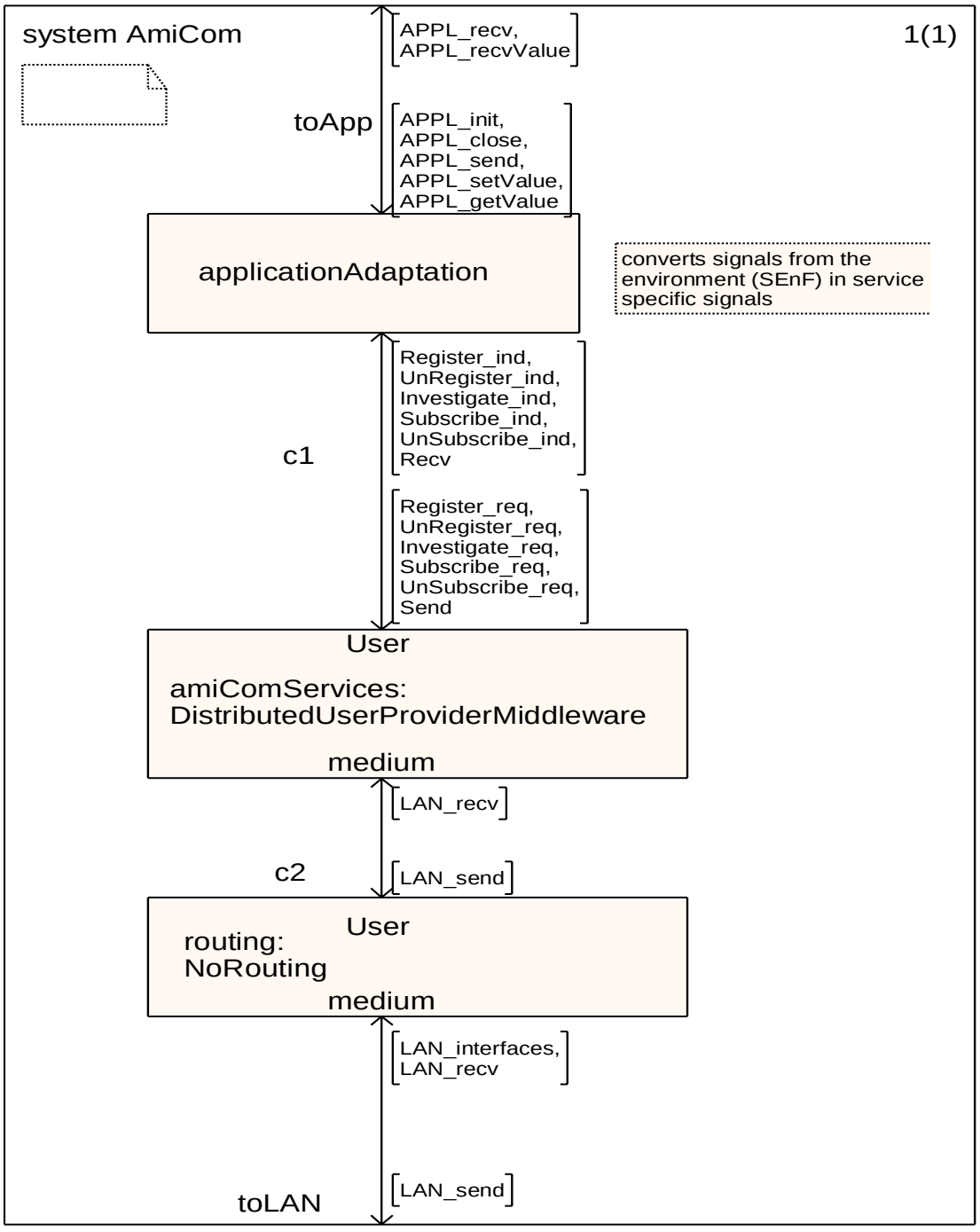
Provided definitions

- Process type ServiceProvider
- Process type ServiceUser
- Process type userProviderManagement

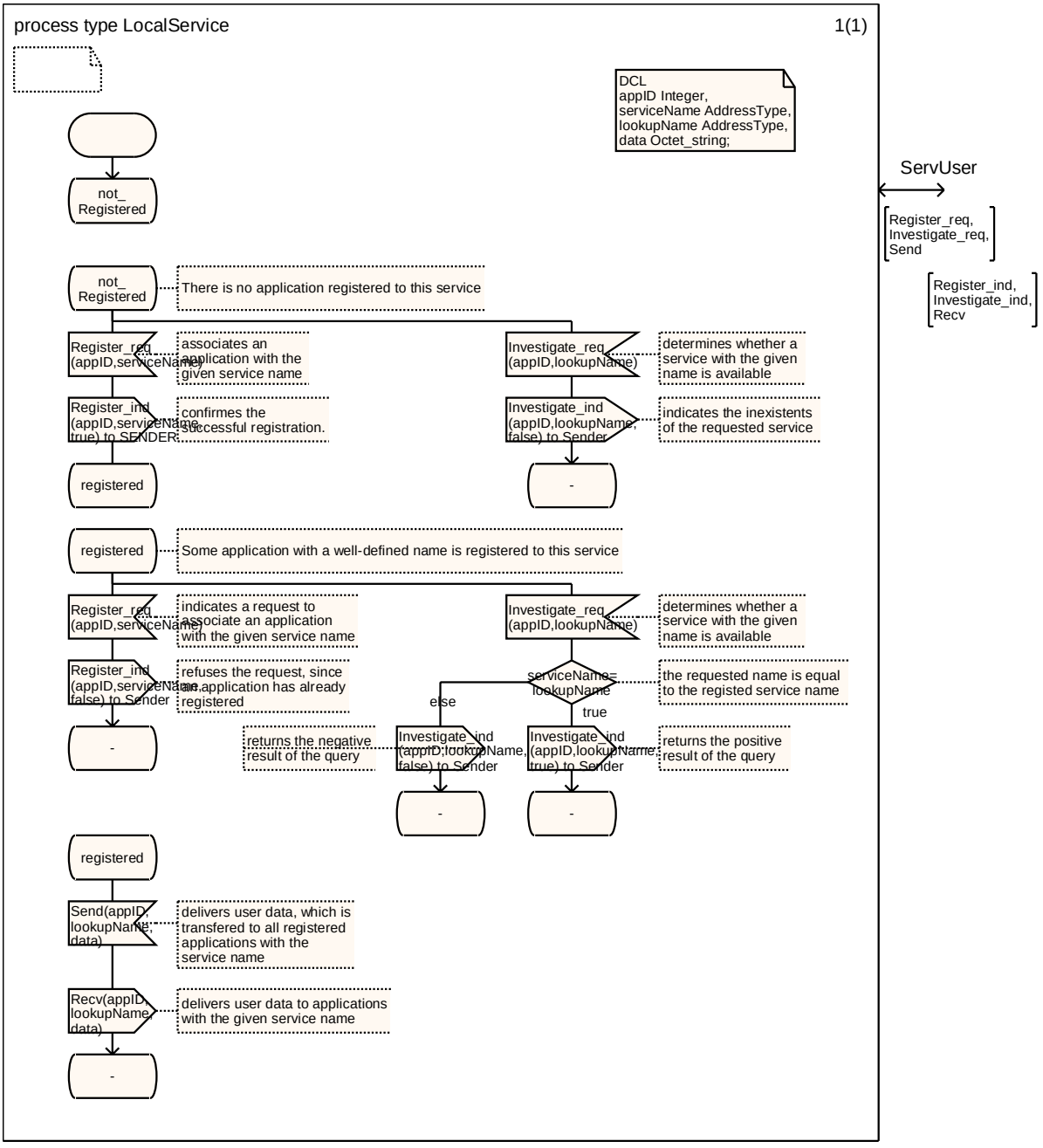
Checklist

– *none* –

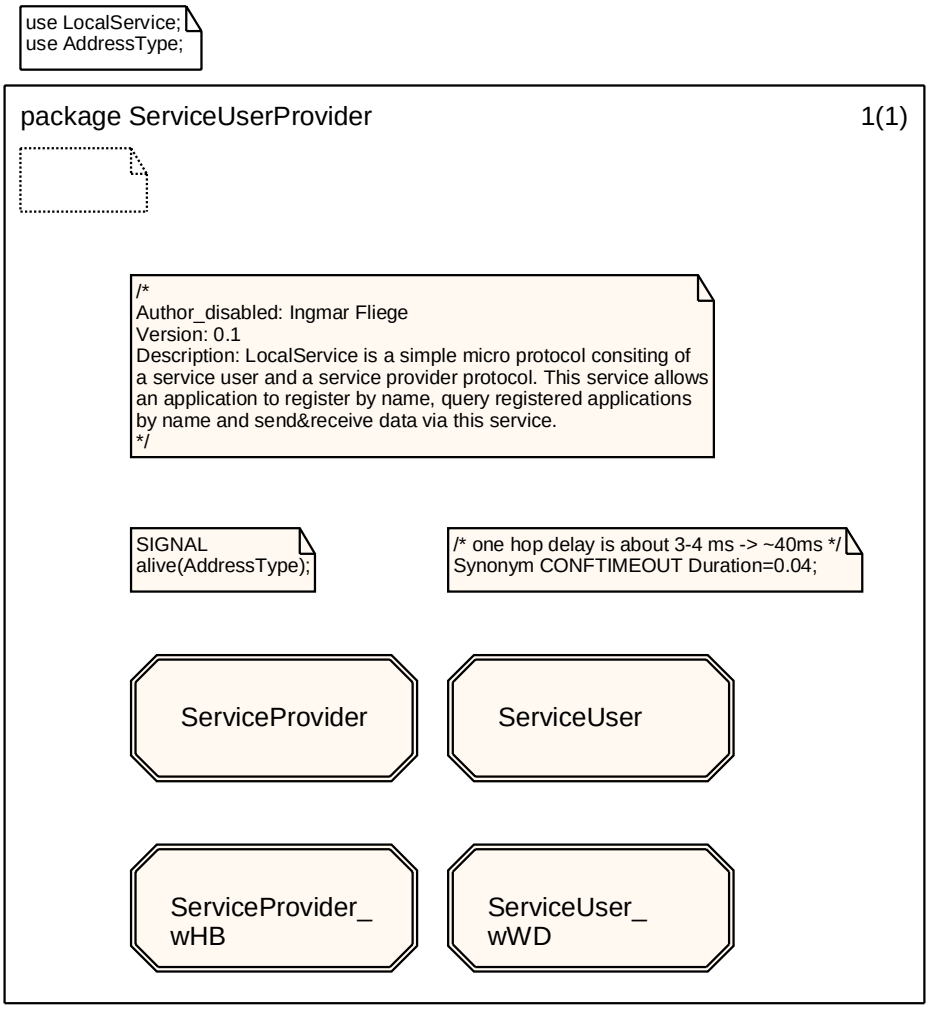
5 Annex A: SDL specification

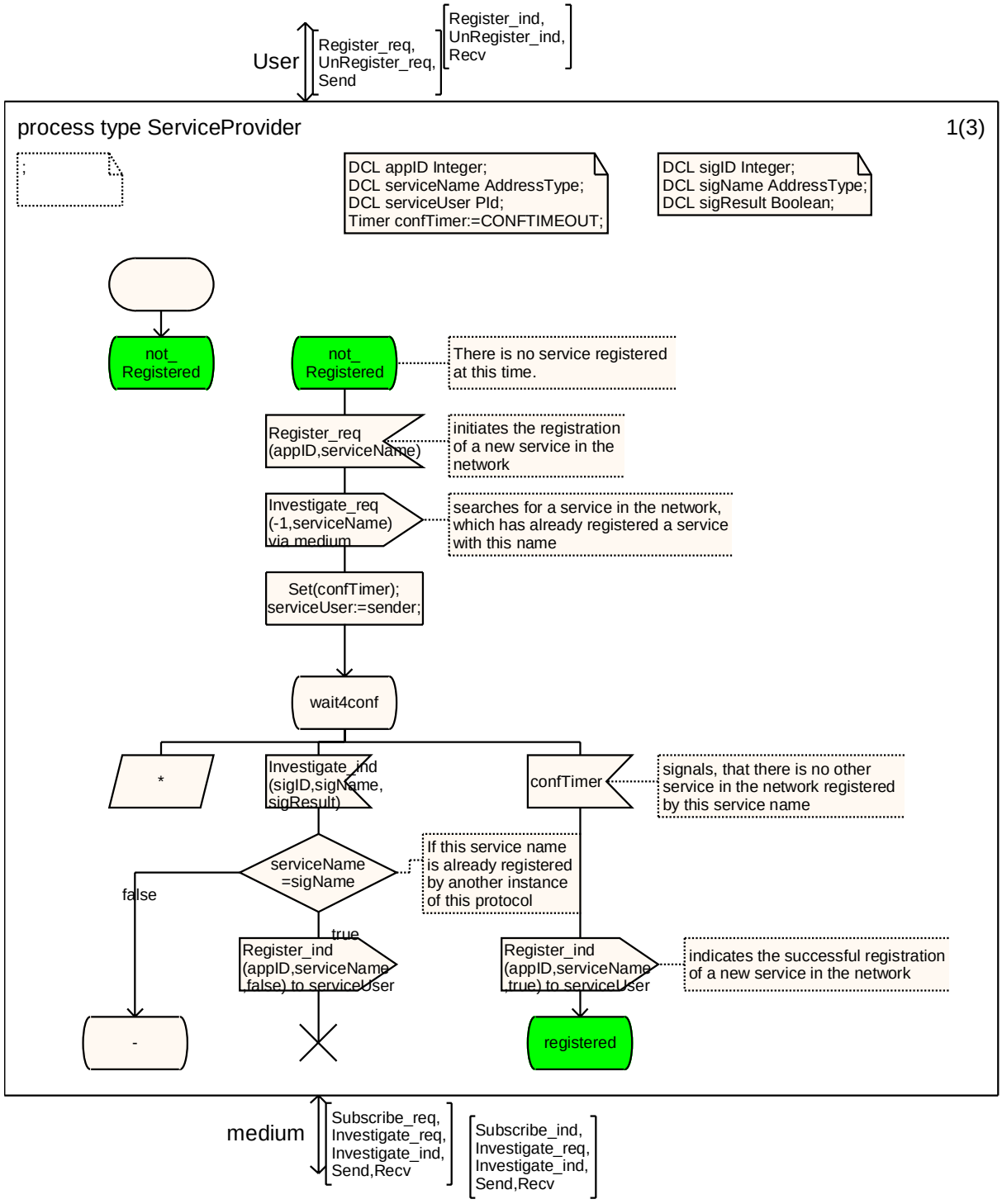


5.1 Service Specification (not distributed)

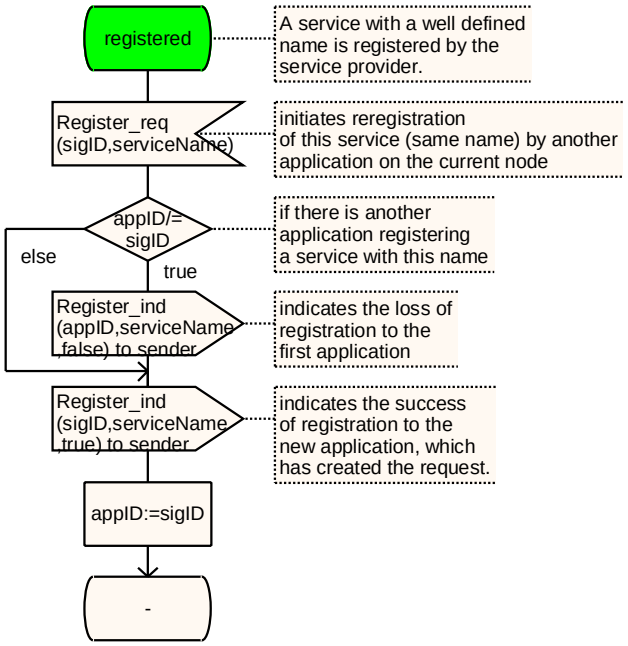


5.2 Service User & Service Provider Middleware

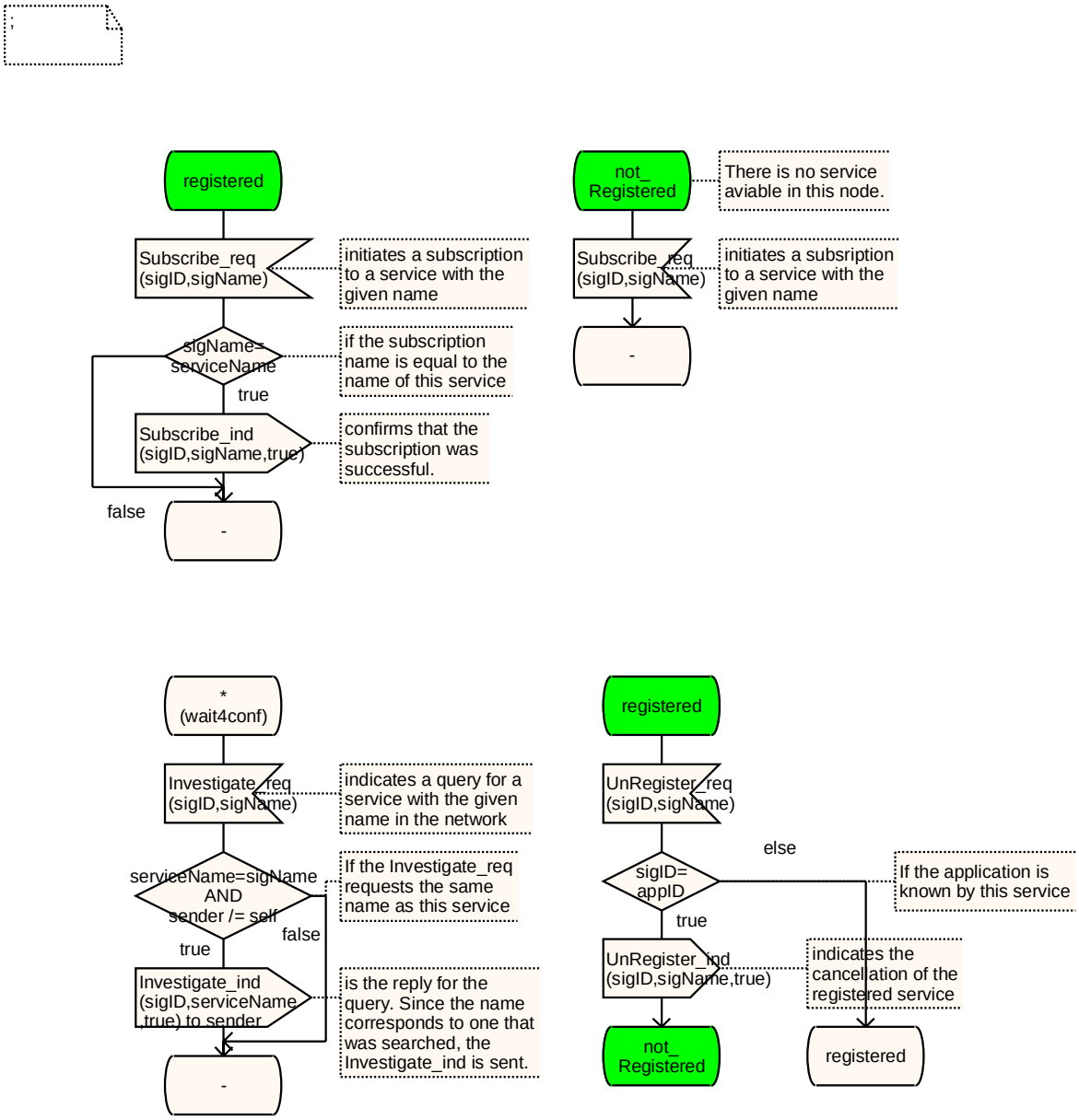


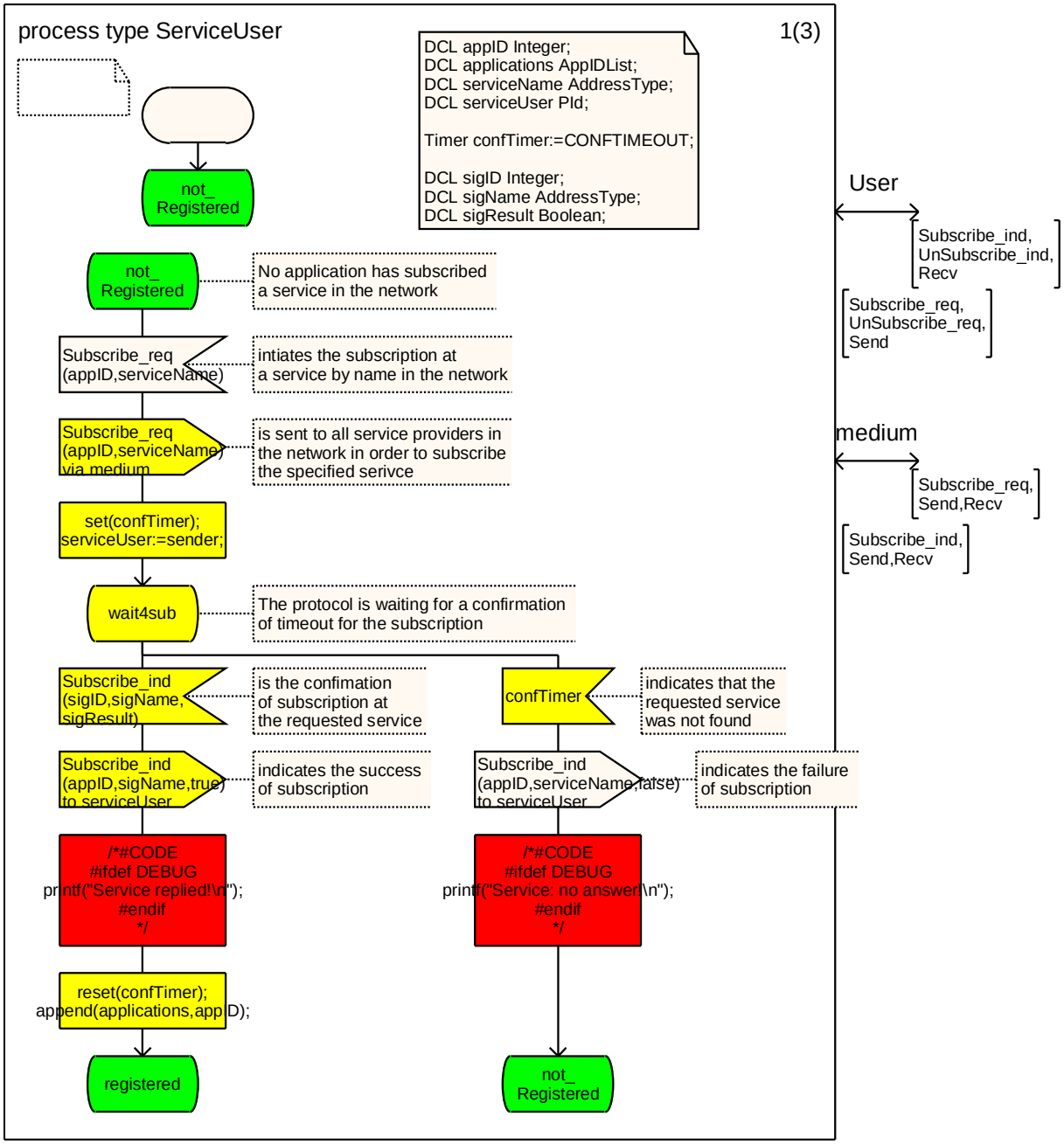


;

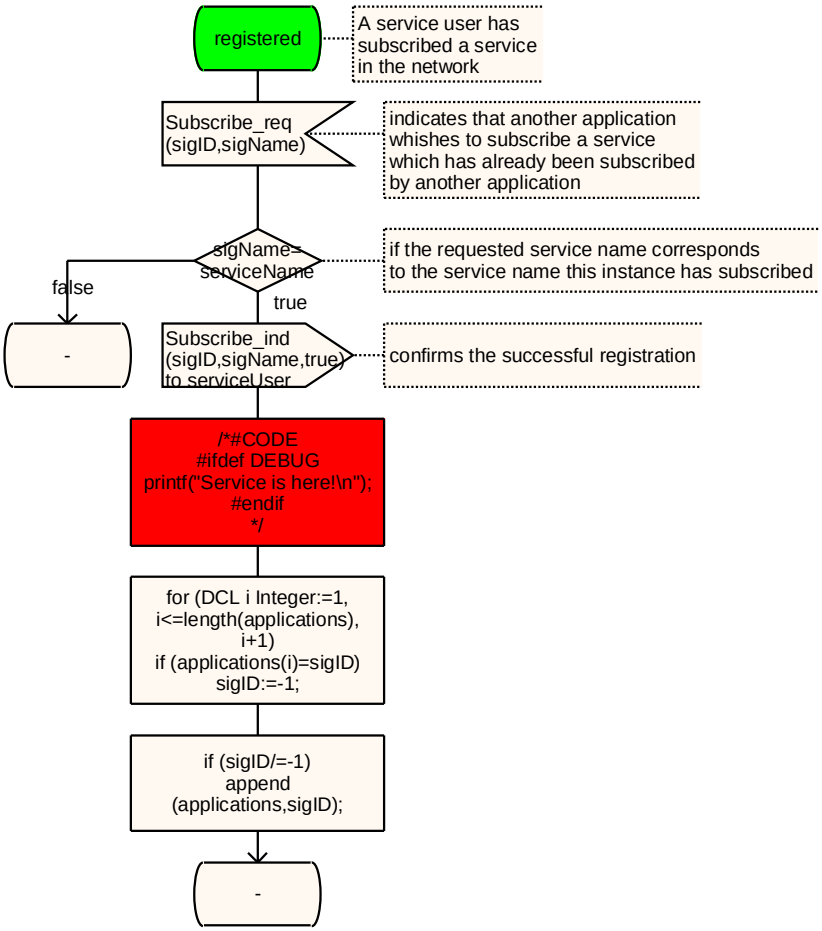

registered
A service with a well defined name is registered by the service provider.
Register_req (sigID,serviceName)
initiates reregistration of this service (same name) by another application on the current node
appID/= sigID
if there is another application registering a service with this name
else
true
Register_ind (appID,serviceName ,false) to sender
indicates the loss of registration to the first application
Register_ind (sigID,serviceName ,true) to sender
indicates the success of registration to the new application, which has created the request.
appID:=sigID
-

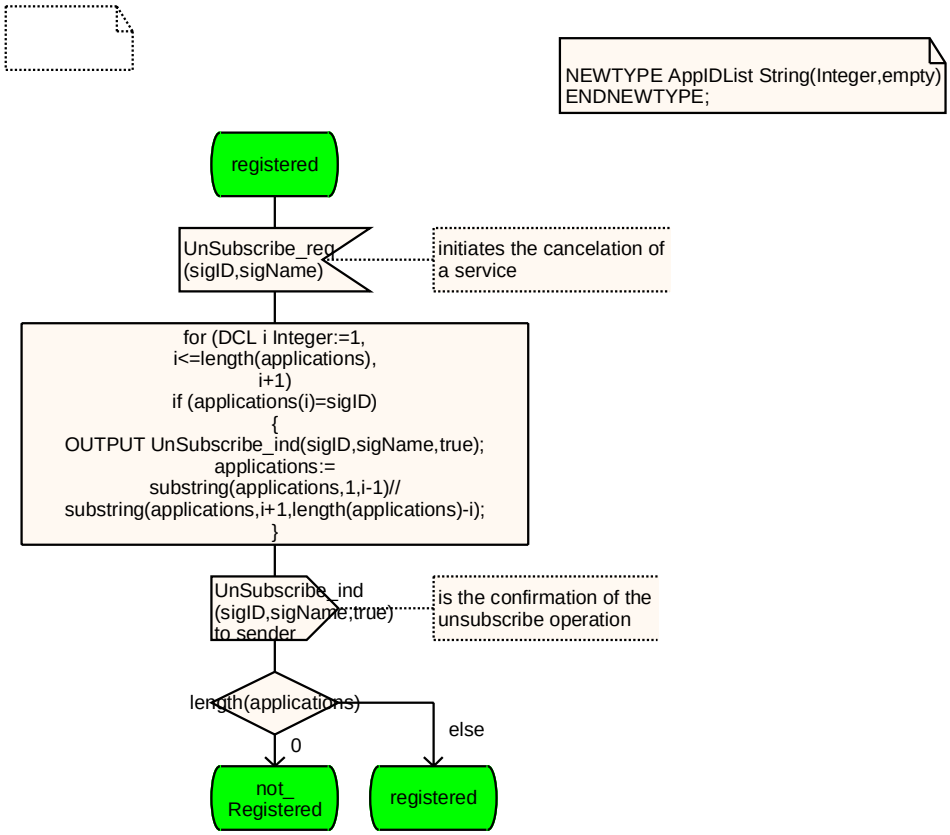
process type ServiceProvider 3(3)

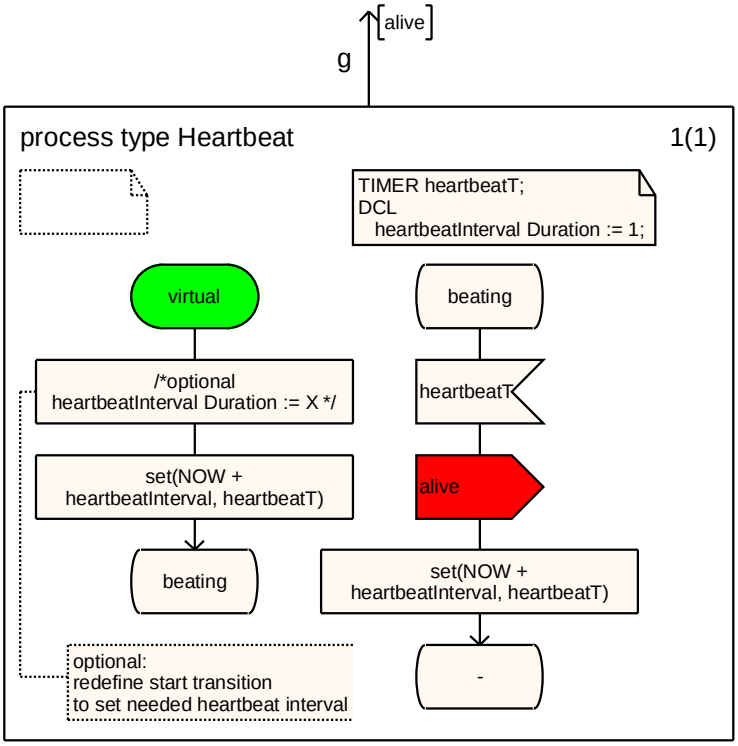


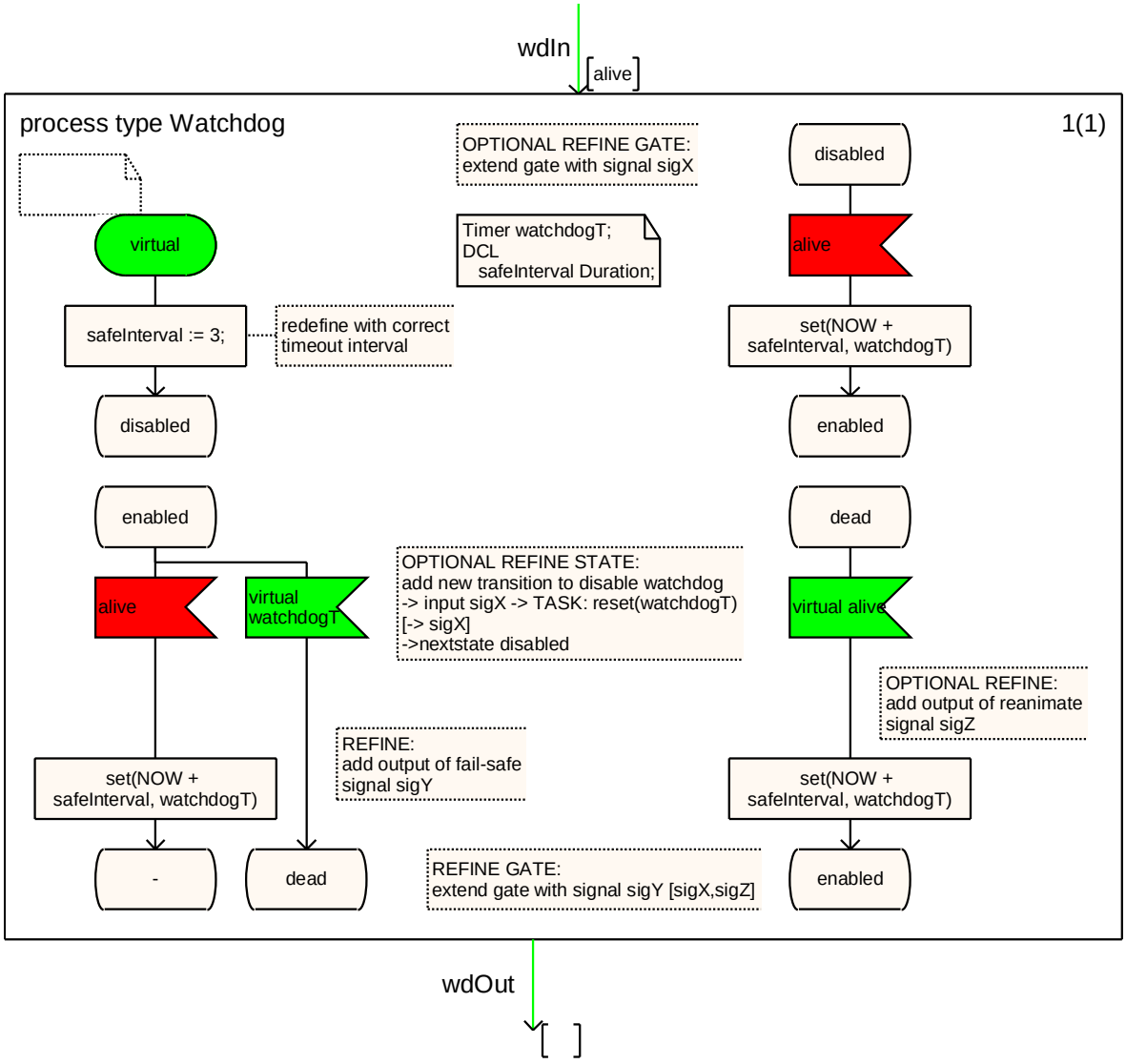

process type ServiceUser
1(3)
DCL appID Integer;
DCL applications AppIDList;
DCL serviceName AddressType;
DCL serviceUser PId;
Timer confTimer:=CONFTIMEOUT;
DCL sigID Integer;
DCL sigName AddressType;
DCL sigResult Boolean;
not_ Registered
not_ Registered
No application has subscribed a service in the network
Subscribe_req (appID,serviceName)
intiates the subscription at a service by name in the network
Subscribe_req (appID,serviceName) via medium
is sent to all service providers in the network in order to subscribe the specified serivce
set(confTimer); serviceUser:=sender;
wait4sub
The protocol is waiting for a confirmation of timeout for the subscription
Subscribe_ind (sigID,sigName, sigResult)
is the confimation of subscription at the requested service
confTimer
indicates that the requested service was not found
Subscribe_ind (appID,sigName,true) to serviceUser
indicates the success of subscription
Subscribe_ind (appID,serviceName,false) to serviceUser
indicates the failure of subscription
/*#CODE #ifdef DEBUG printf("Service replied!\n"); #endif */
/*#CODE #ifdef DEBUG printf("Service: no answer!\n"); #endif */
reset(confTimer); append(applications,appID);
registered
not_ Registered
User
Subscribe_ind, UnSubscribe_ind, Recv
Subscribe_req, UnSubscribe_req, Send
medium
Subscribe_req, Send,Recv
Subscribe_ind, Send,Recv

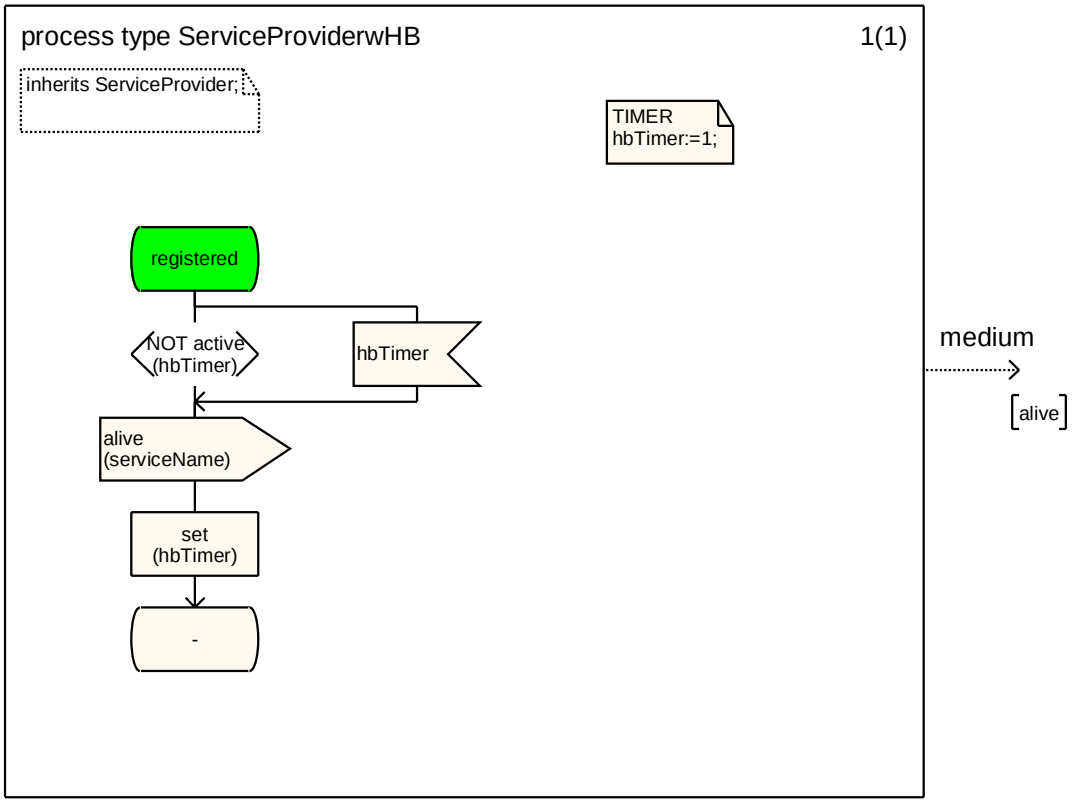
process type ServiceUser 2(3)

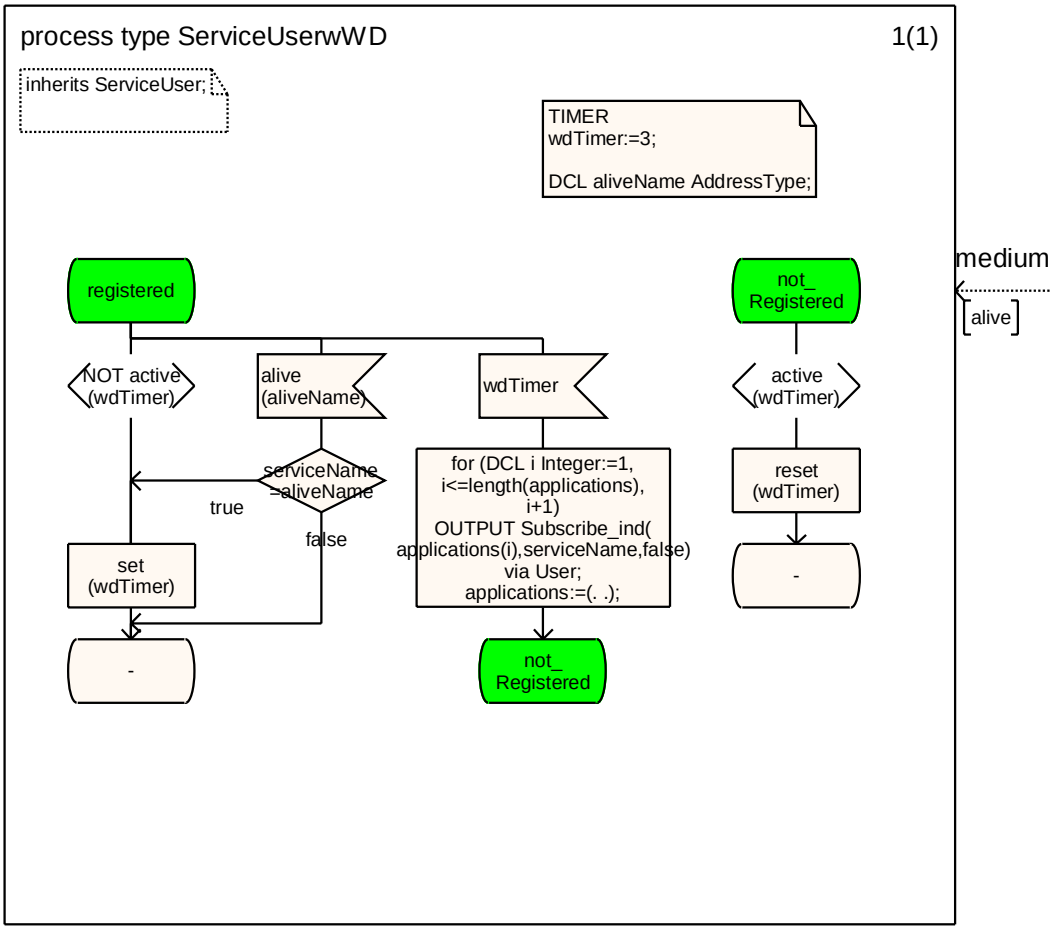



NEWTYPE AppIDList String(Integer,empty)
ENDNEWTYPE;
registered
UnSubscribe_req
(sigID,sigName)
initiates the cancelation of
a service
for (DCL i Integer:=1,
i<=length(applications),
i+1)
if (applications(i)=sigID)
{
OUTPUT UnSubscribe_ind(sigID,sigName,true);
applications:=
substring(applications,1,i-1)//
substring(applications,i+1,length(applications)-i);
}
UnSubscribe_ind
(sigID,sigName,true)
to sender
is the confirmation of the
unsubscribe operation
length(applications)
0
else
not_
Registered
registered

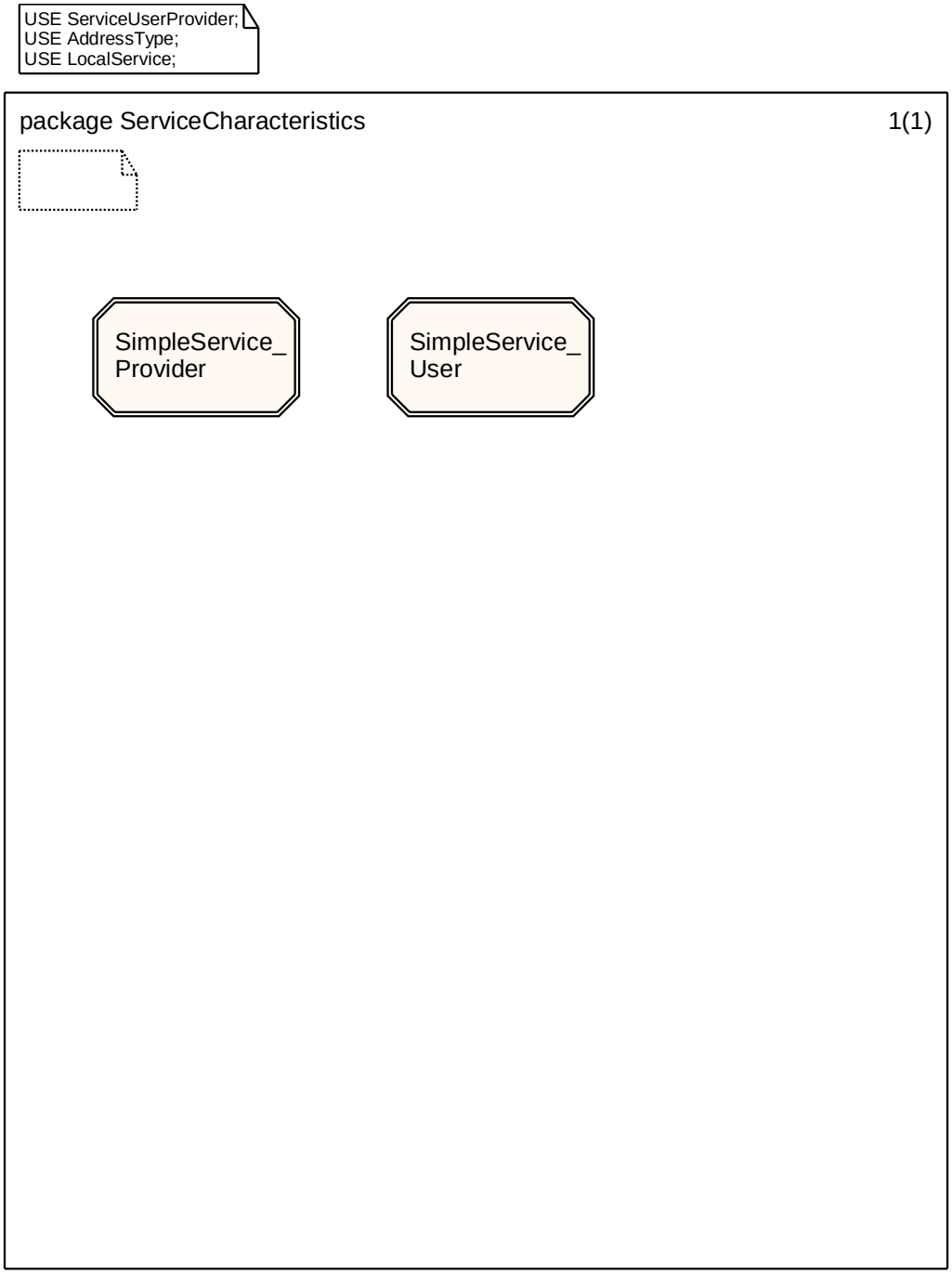



wdIn
alive
process type Watchdog
1(1)
OPTIONAL REFINE GATE:
extend gate with signal sigX
Timer watchdogT;
DCL
safeInterval Duration;
virtual
safeInterval := 3;
redefine with correct
timeout interval
disabled
enabled
alive
virtual
watchdogT
OPTIONAL REFINE STATE:
add new transition to disable watchdog
-> input sigX -> TASK: reset(watchdogT)
[-> sigX]
->nextstate disabled
REFINE:
add output of fail-safe
signal sigY
set(NOW +
safeInterval, watchdogT)
-
dead
REFINE GATE:
extend gate with signal sigY [sigX,sigZ]
disabled
alive
set(NOW +
safeInterval, watchdogT)
enabled
dead
virtual alive
OPTIONAL REFINE:
add output of reanimate
signal sigZ
set(NOW +
safeInterval, watchdogT)
enabled
wdOut
[ ]

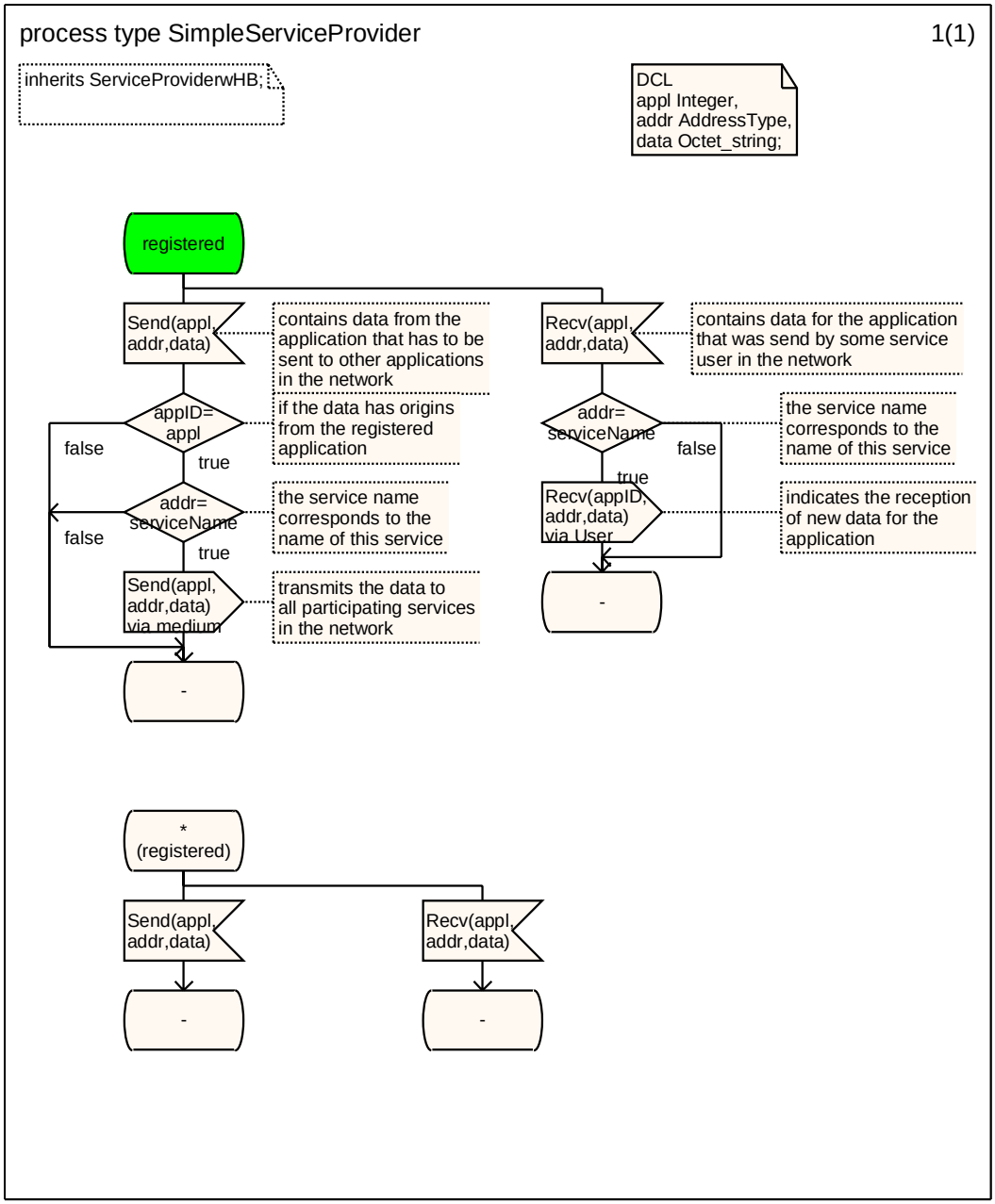


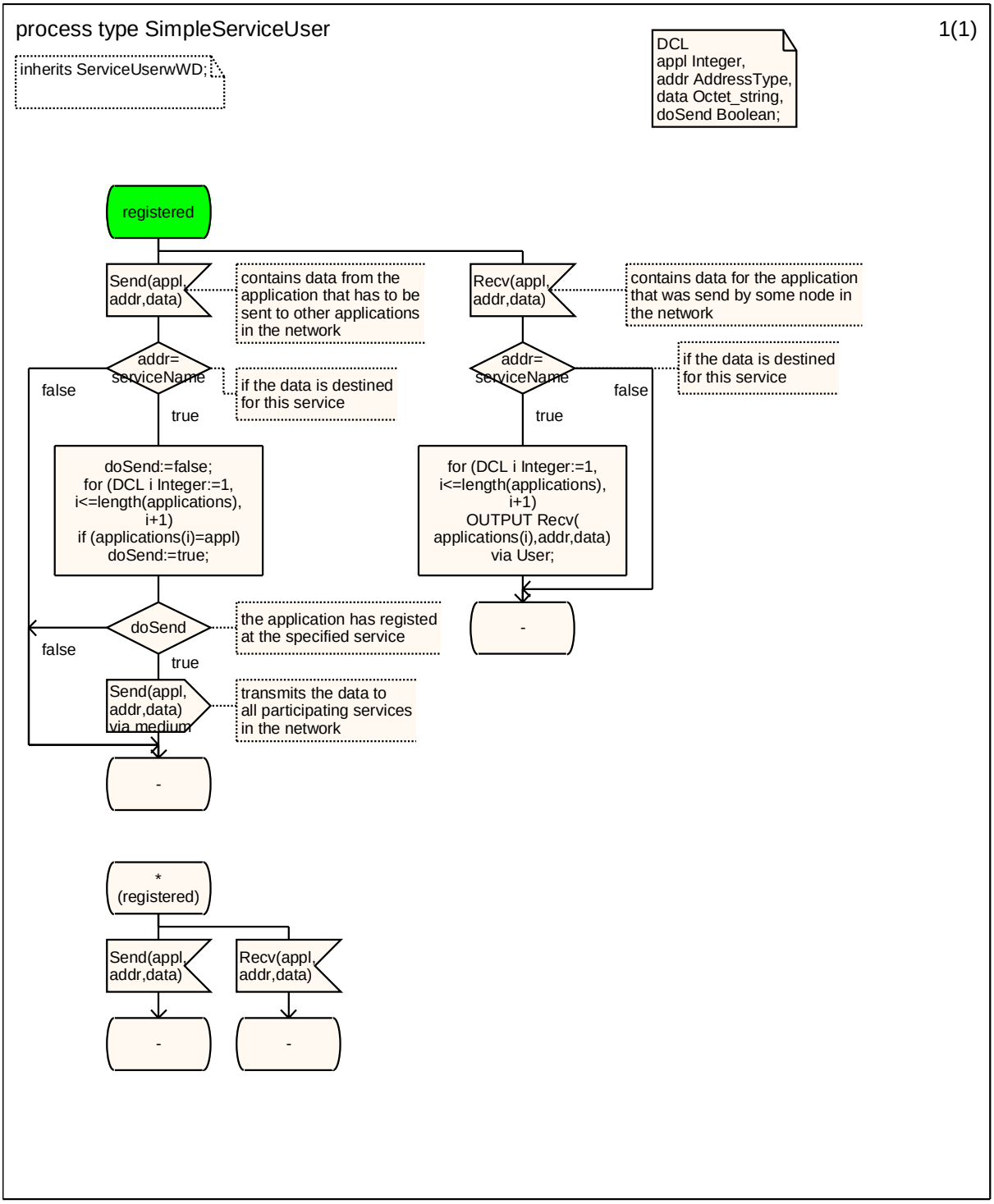

process type ServiceUserwWD
1(1)
inherits ServiceUser;
TIMER
wdTimer:=3;
DCL aliveName AddressType;
medium
alive
registered
not_
Registered
NOT active
(wdTimer)
alive
(aliveName)
wdTimer
active
(wdTimer)
serviceName
=aliveName
true
false
for (DCL i Integer:=1,
i<=length(applications),
i+1)
OUTPUT Subscribe_ind(
applications(i),serviceName,false)
via User;
applications:=(. .);
reset
(wdTimer)
set
(wdTimer)
-
-
not_
Registered

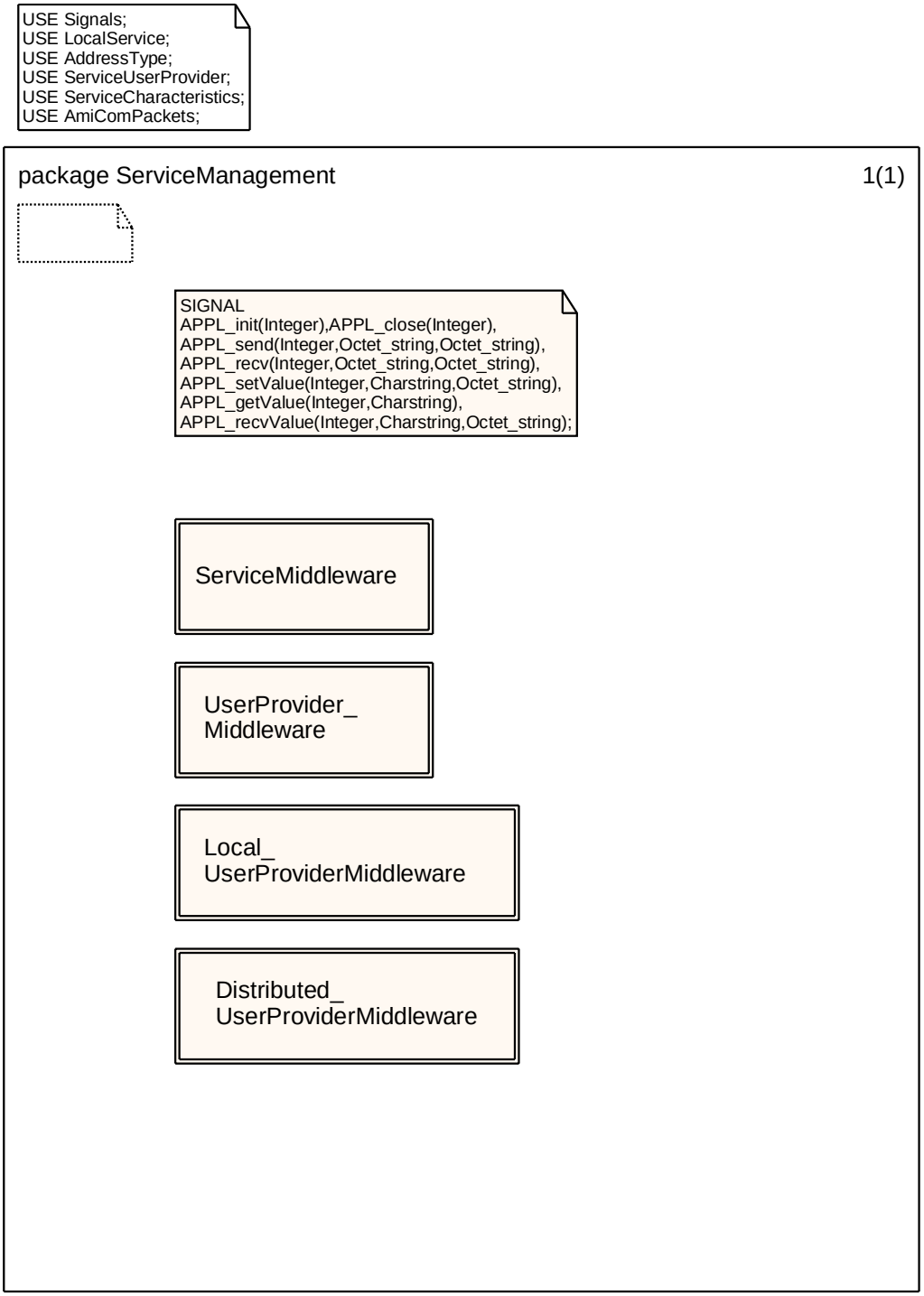
5.3 Service User & Service Provider Characteristics

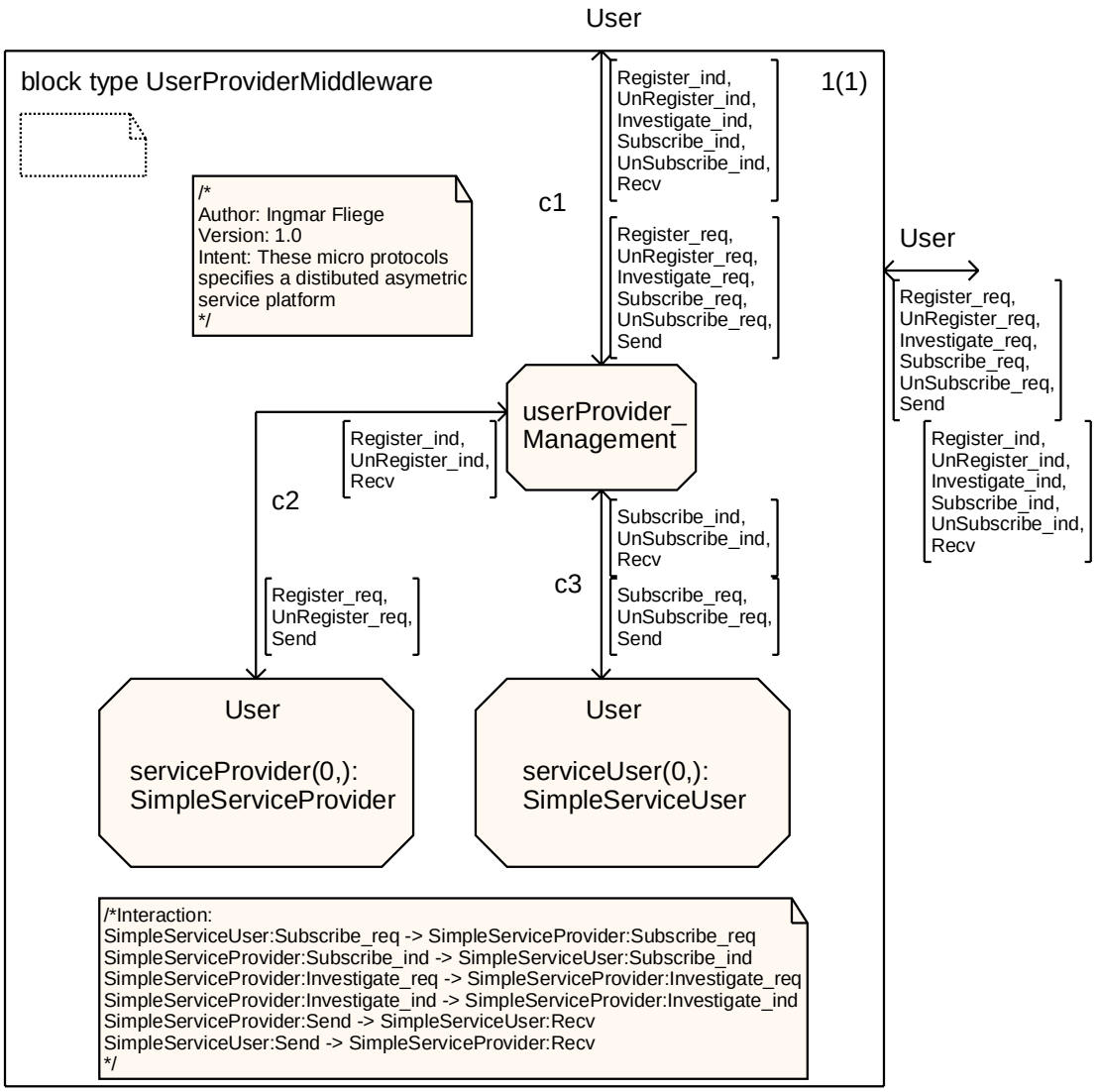


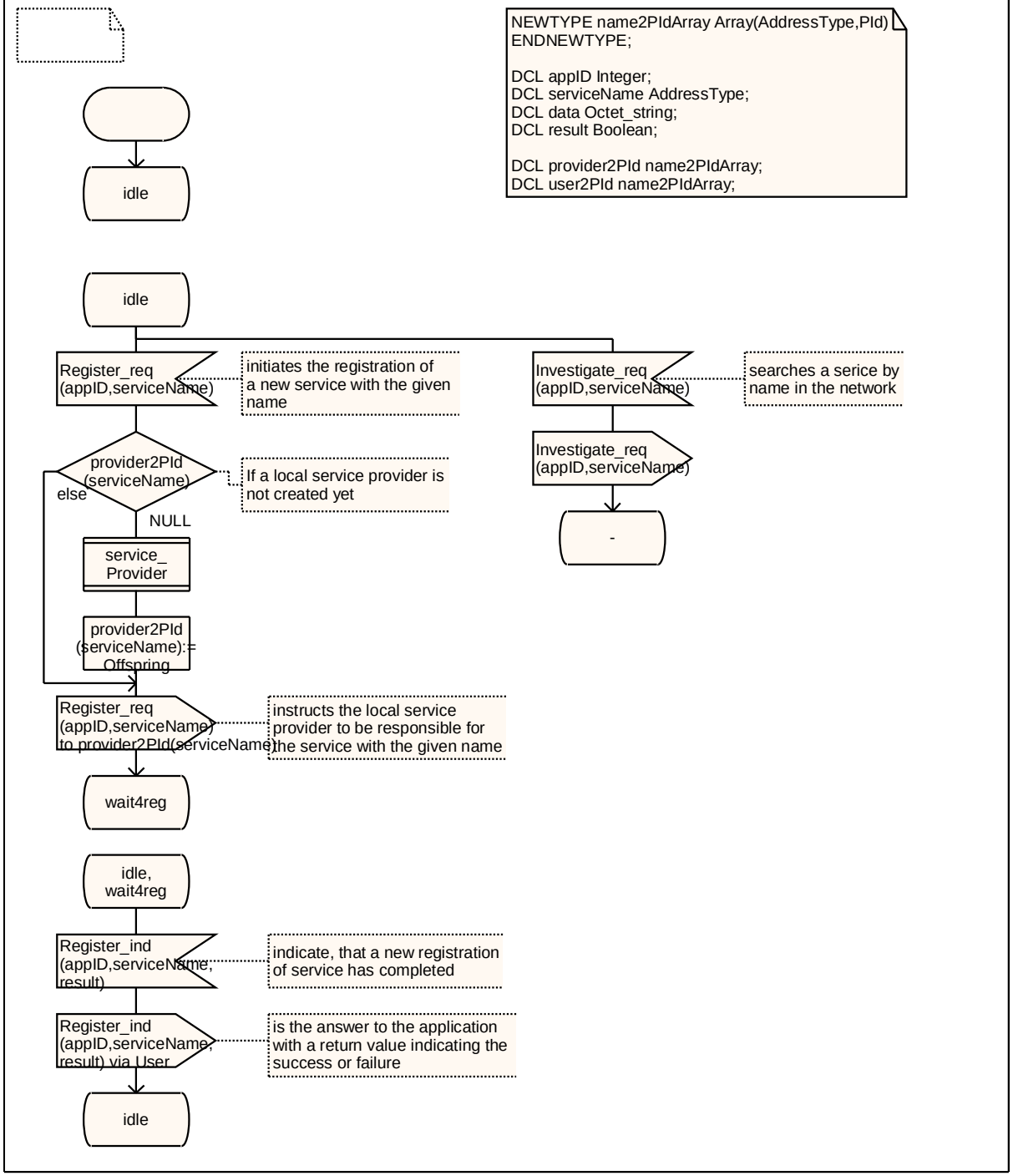
process type SimpleServiceProvider 1(1)



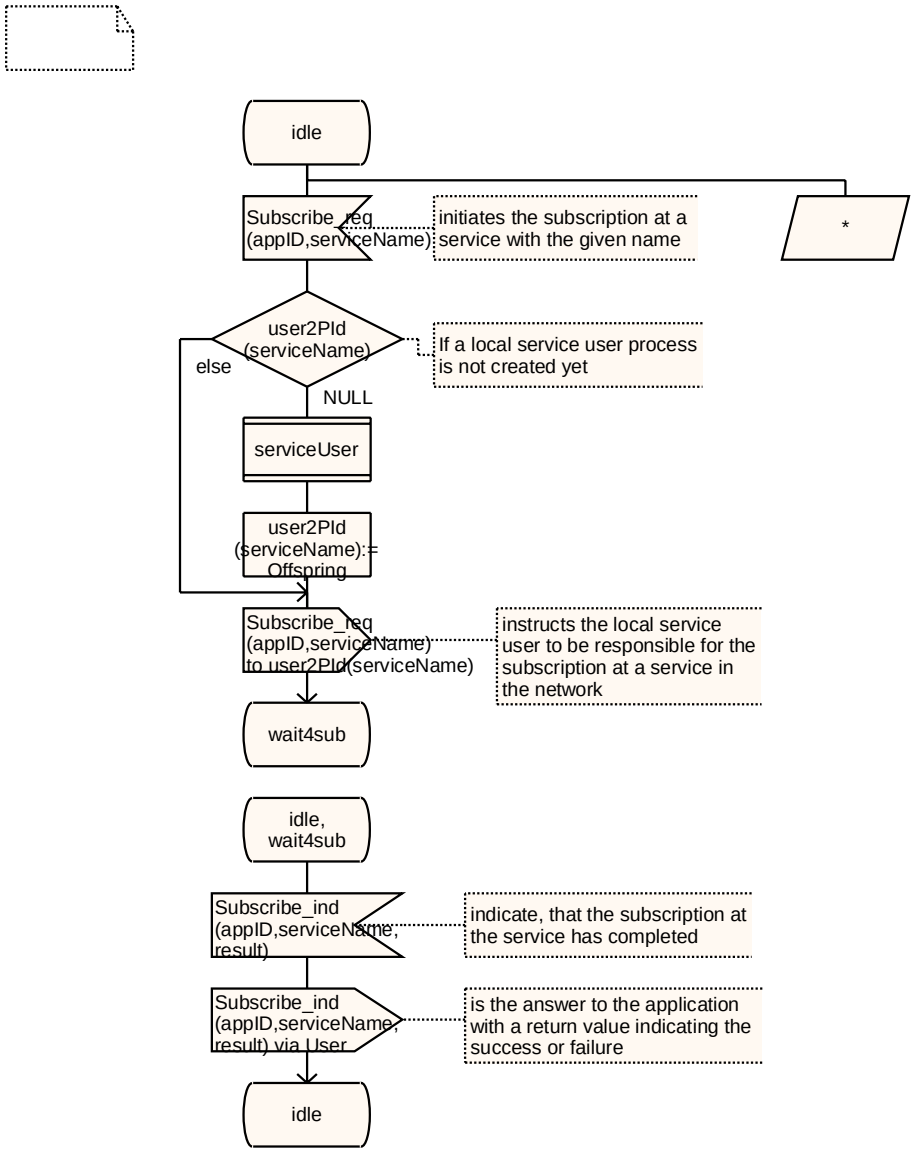


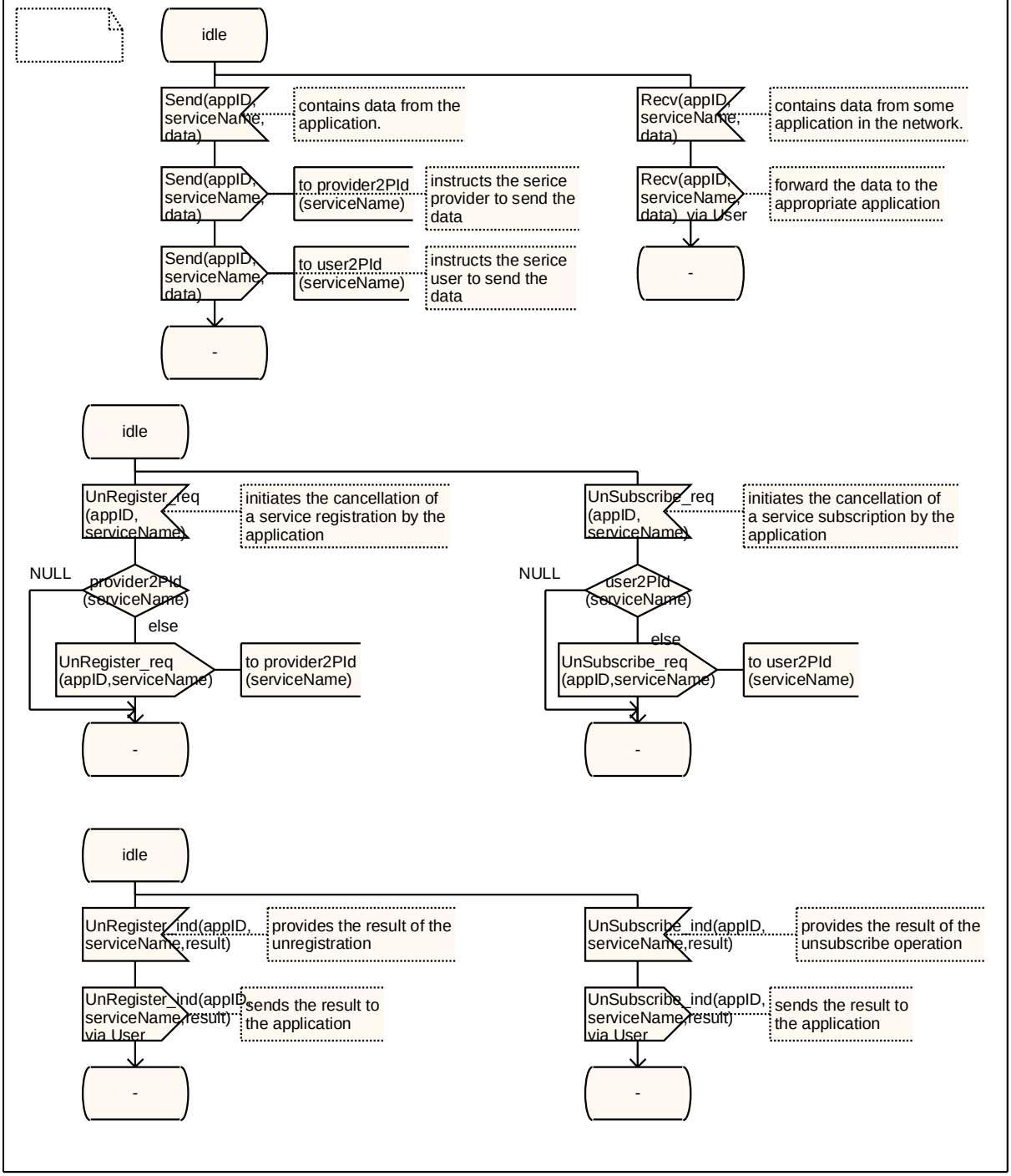

USE Signals;
USE LocalService;
USE AddressType;
USE ServiceUserProvider;
USE ServiceCharacteristics;
USE AmiComPackets;
package ServiceManagement
1(1)
SIGNAL
APPL_init(Integer),APPL_close(Integer),
APPL_send(Integer,Octet_string,Octet_string),
APPL_recv(Integer,Octet_string,Octet_string),
APPL_setValue(Integer,Charstring,Octet_string),
APPL_getValue(Integer,Charstring),
APPL_recvValue(Integer,Charstring,Octet_string);
ServiceMiddleware
UserProvider_
Middleware
Local_
UserProviderMiddleware
Distributed_
UserProviderMiddleware

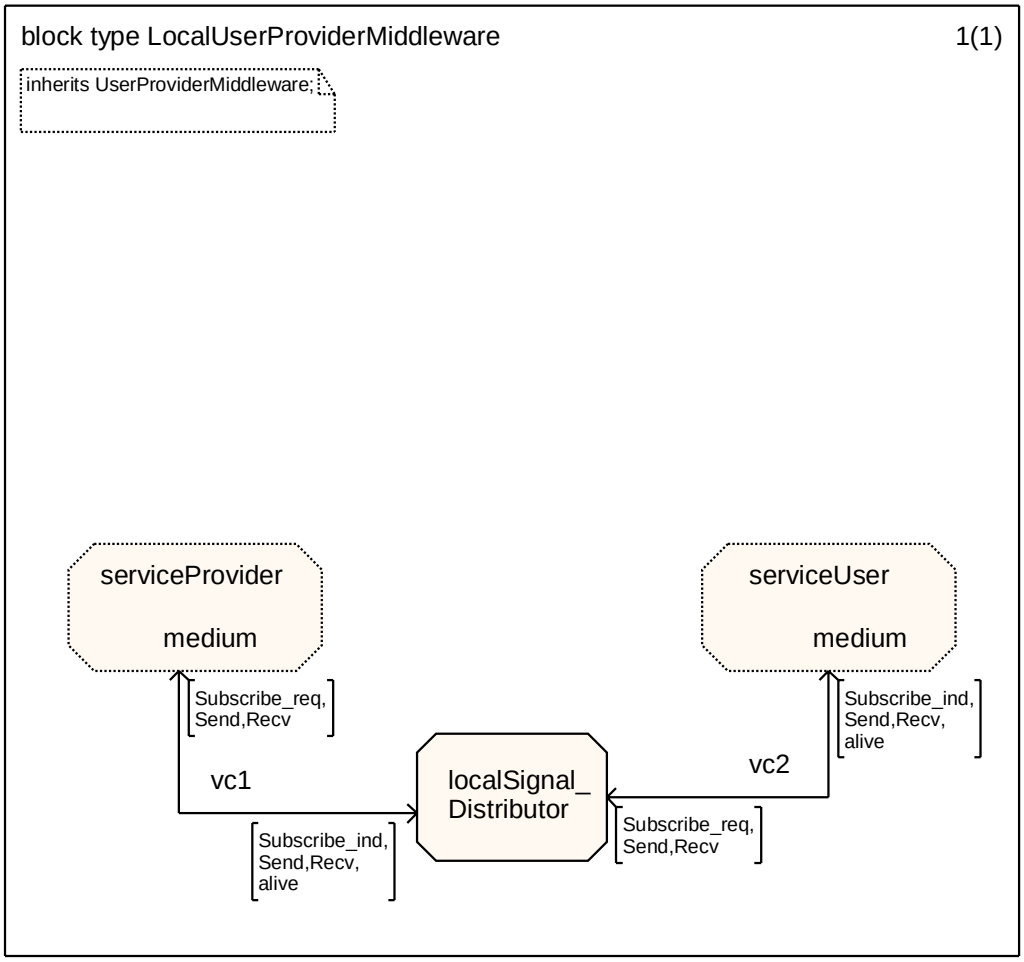



NEWTYPE name2PIdArray Array(AddressType,PId)
ENDNEWTYPE;
DCL appID Integer;
DCL serviceName AddressType;
DCL data Octet_string;
DCL result Boolean;
DCL provider2PId name2PIdArray;
DCL user2PId name2PIdArray;
idle
idle
Register_req (appID,serviceName)
initiates the registration of a new service with the given name
Investigate_req (appID,serviceName)
searches a serice by name in the network
provider2PId (serviceName)
else
If a local service provider is not created yet
NULL
Investigate_req (appID,serviceName)
-
service_ Provider
provider2PId (serviceName):= Offspring
Register_req (appID,serviceName) to provider2PId(serviceName)
instructs the local service provider to be responsible for the service with the given name
wait4reg
idle, wait4reg
Register_ind (appID,serviceName, result)
indicate, that a new registration of service has completed
Register_ind (appID,serviceName, result) via User
is the answer to the application with a return value indicating the success or failure
idle

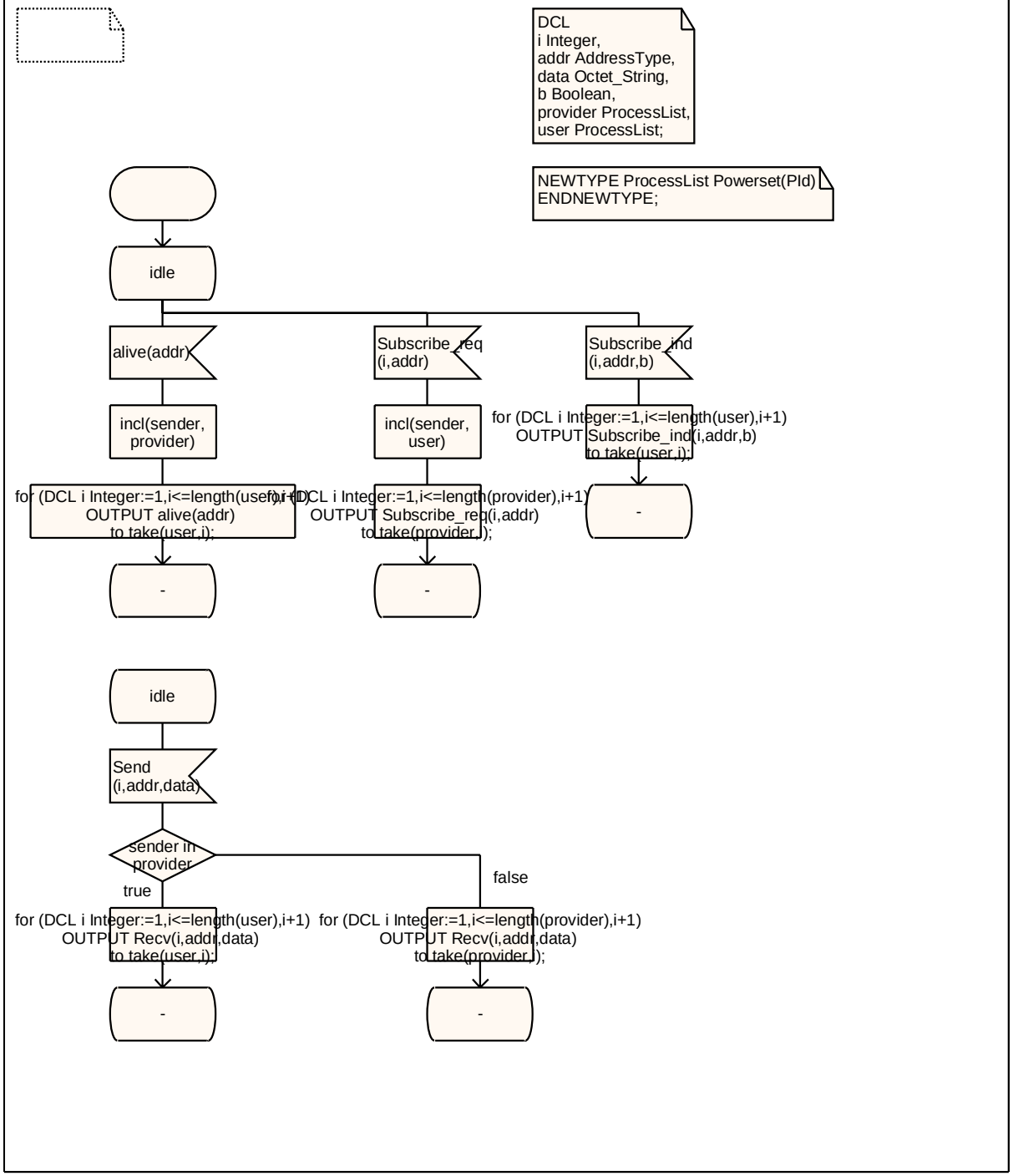
process userProviderManagement 2(3)

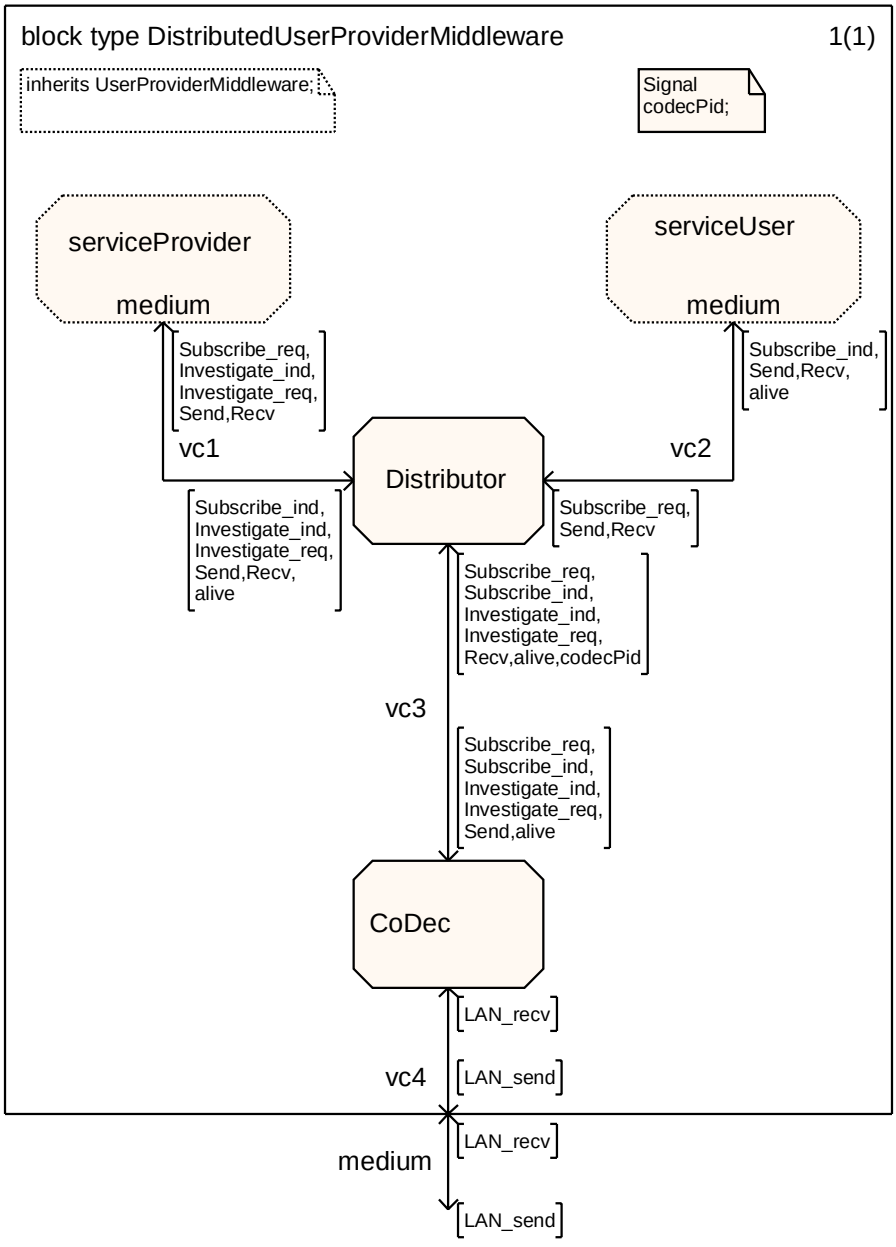


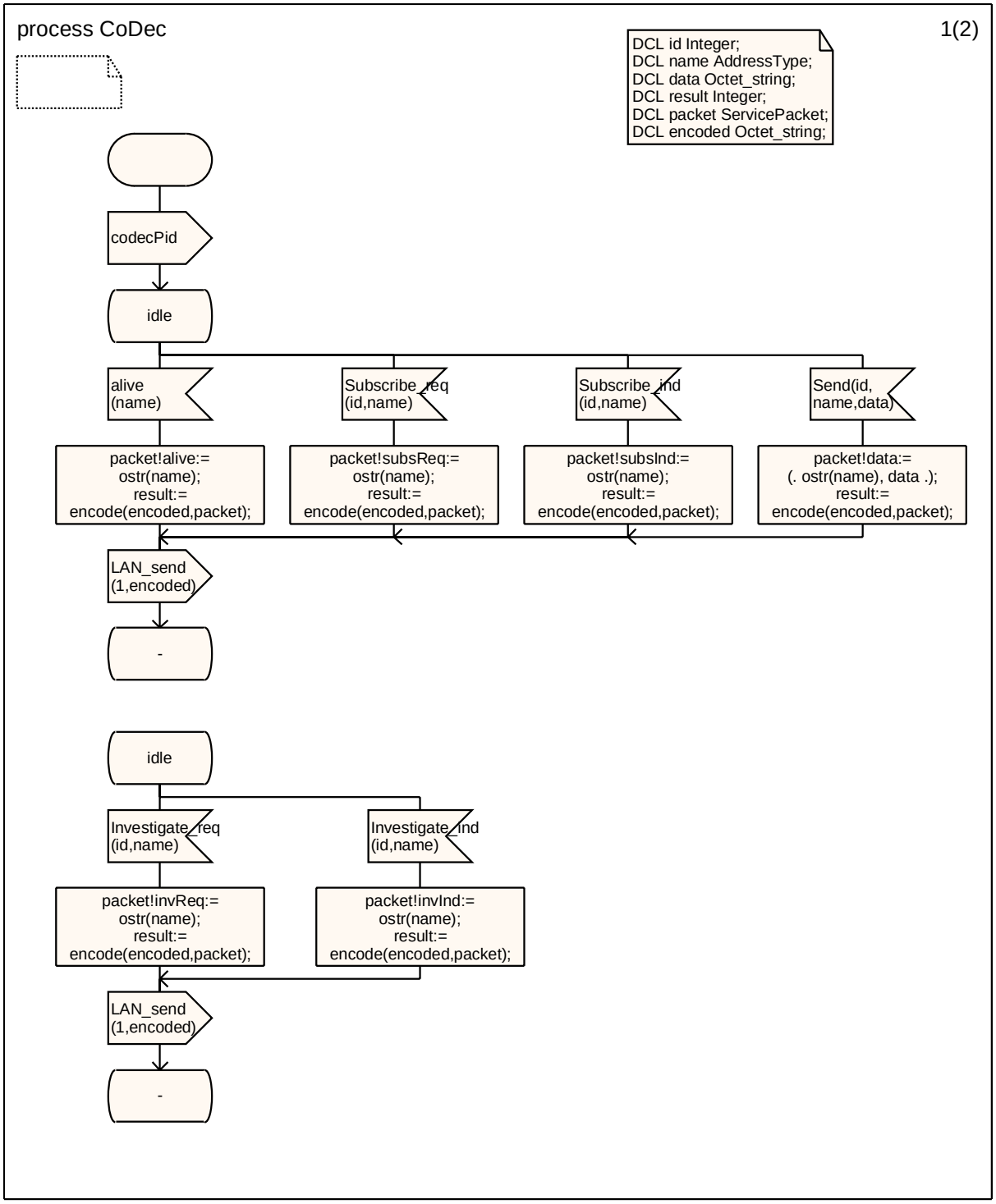

idle
Send(appID, serviceName, data)
contains data from the application.
Send(appID, serviceName, data)
to provider2PId (serviceName)
instructs the serice provider to send the data
Send(appID, serviceName, data)
to user2PId (serviceName)
instructs the serice user to send the data
-
Recv(appID, serviceName, data)
contains data from some application in the network.
Recv(appID, serviceName, data) via User
forward the data to the appropriate application
-
idle
UnRegister_req (appID, serviceName)
initiates the cancellation of a service registration by the application
NULL
provider2PId (serviceName)
else
UnRegister_req (appID,serviceName)
to provider2PId (serviceName)
-
UnSubscribe_req (appID, serviceName)
initiates the cancellation of a service subscription by the application
NULL
user2PId (serviceName)
else
UnSubscribe_req (appID,serviceName)
to user2PId (serviceName)
-
idle
UnRegister_ind(appID, serviceName,result)
provides the result of the unregistration
UnRegister_ind(appID, serviceName,result) via User
sends the result to the application
-
UnSubscribe_ind(appID, serviceName,result)
provides the result of the unsubscribe operation
UnSubscribe_ind(appID, serviceName,result) via User
sends the result to the application
-



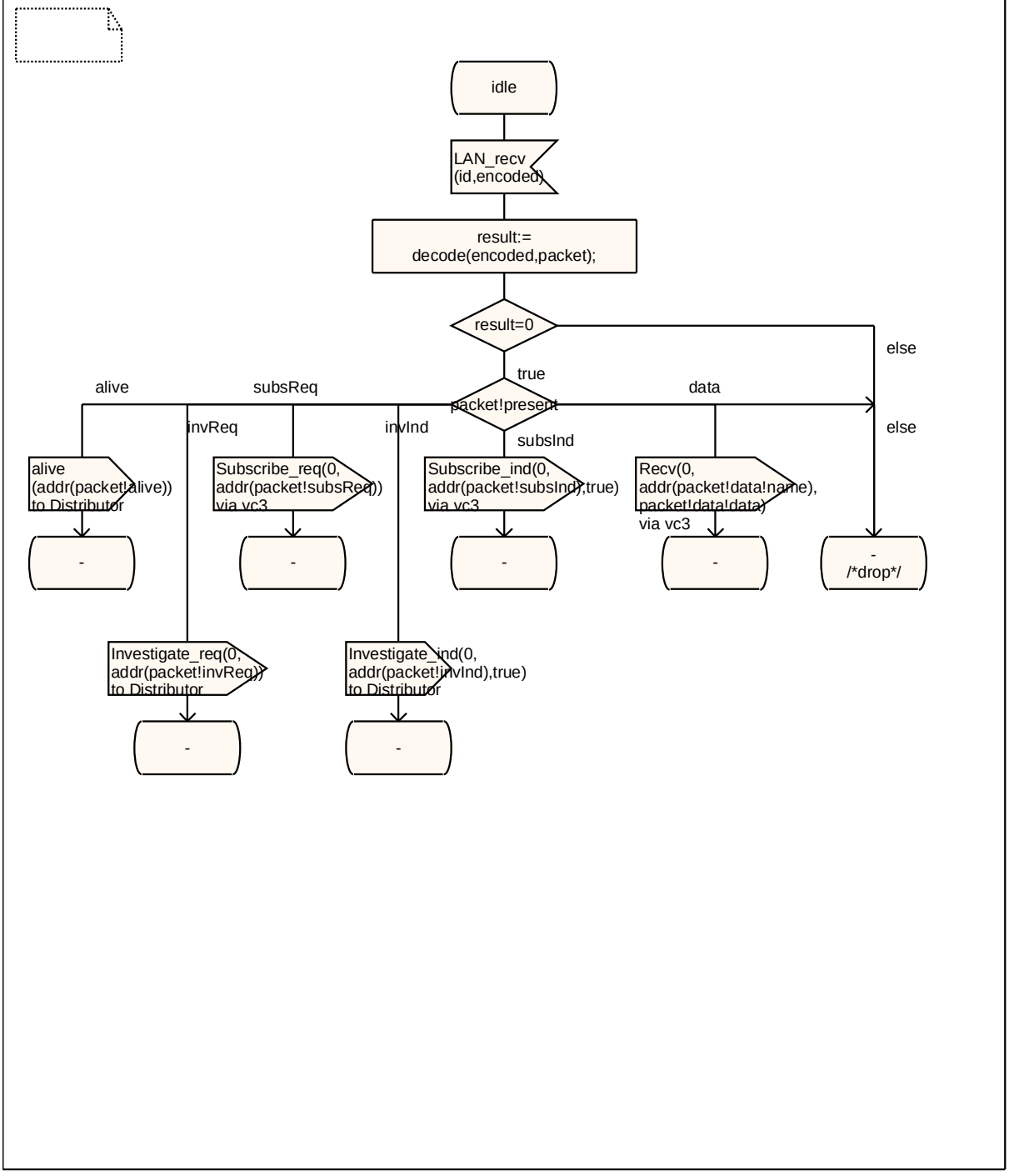
process localSignalDistributor 1(1)



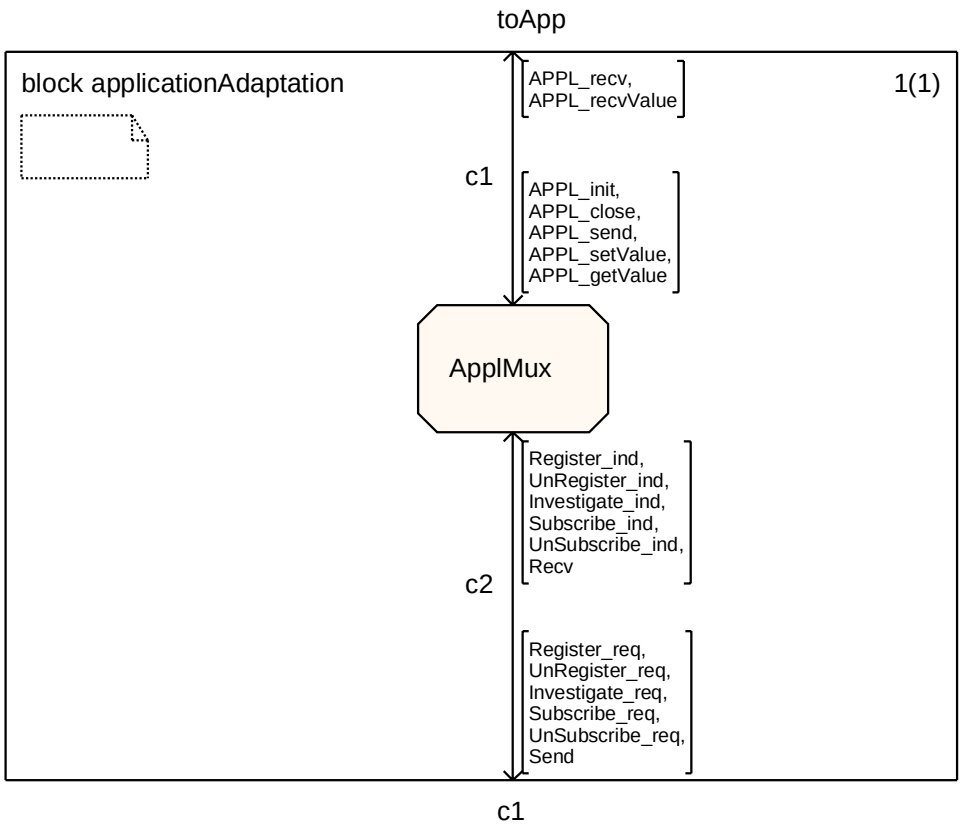


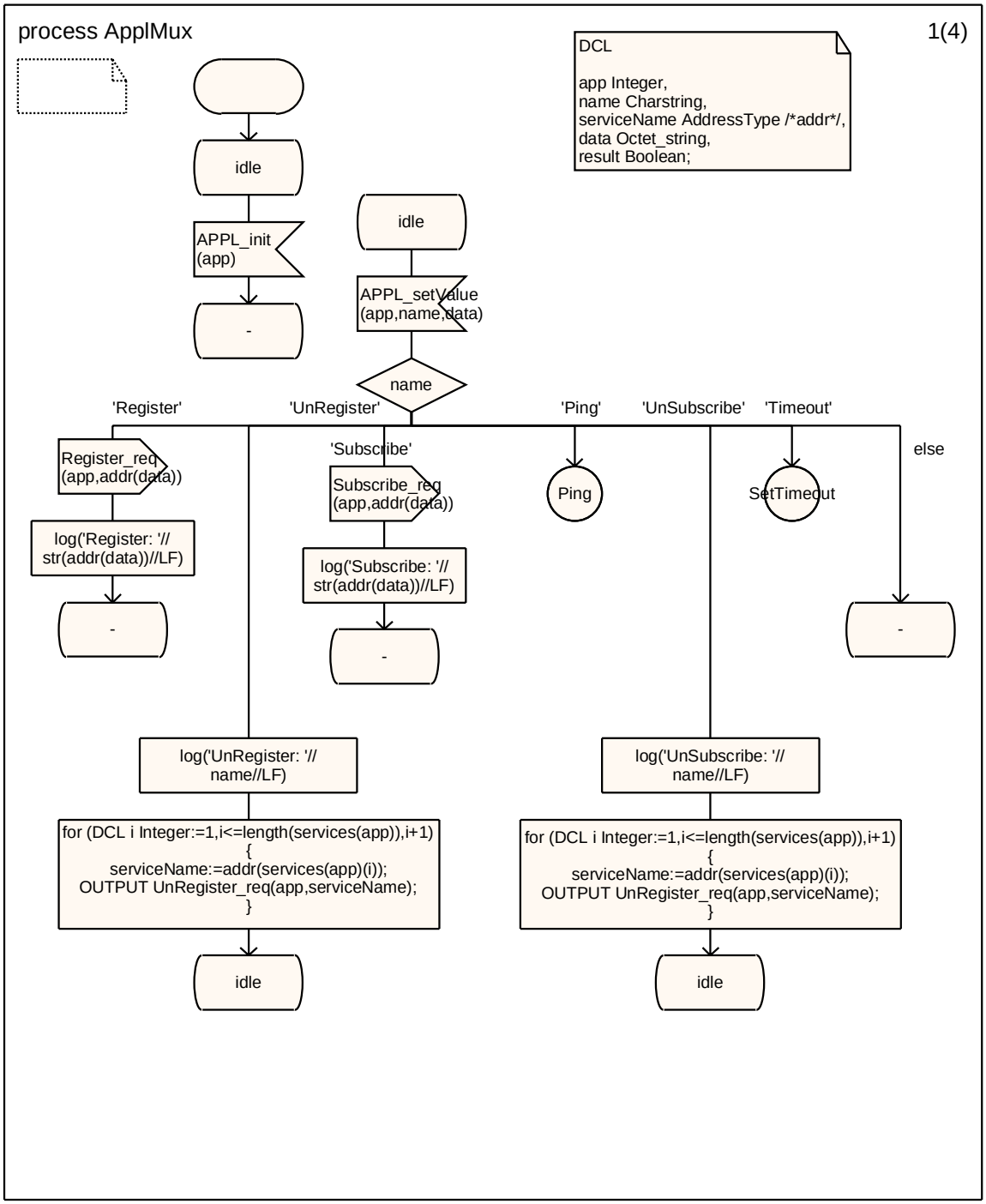

process CoDec
1(2)
DCL id Integer;
DCL name AddressType;
DCL data Octet_string;
DCL result Integer;
DCL packet ServicePacket;
DCL encoded Octet_string;
codecPid
idle
alive
(name)
Subscribe_req
(id,name)
Subscribe_ind
(id,name)
Send(id,
name,data)
packet!alive:=
ostr(name);
result:=
encode(encoded,packet);
packet!subsReq:=
ostr(name);
result:=
encode(encoded,packet);
packet!subsInd:=
ostr(name);
result:=
encode(encoded,packet);
packet!data:=
(. ostr(name), data .);
result:=
encode(encoded,packet);
LAN_send
(1,encoded)
-
idle
Investigate_req
(id,name)
Investigate_ind
(id,name)
packet!invReq:=
ostr(name);
result:=
encode(encoded,packet);
packet!invInd:=
ostr(name);
result:=
encode(encoded,packet);
LAN_send
(1,encoded)
-

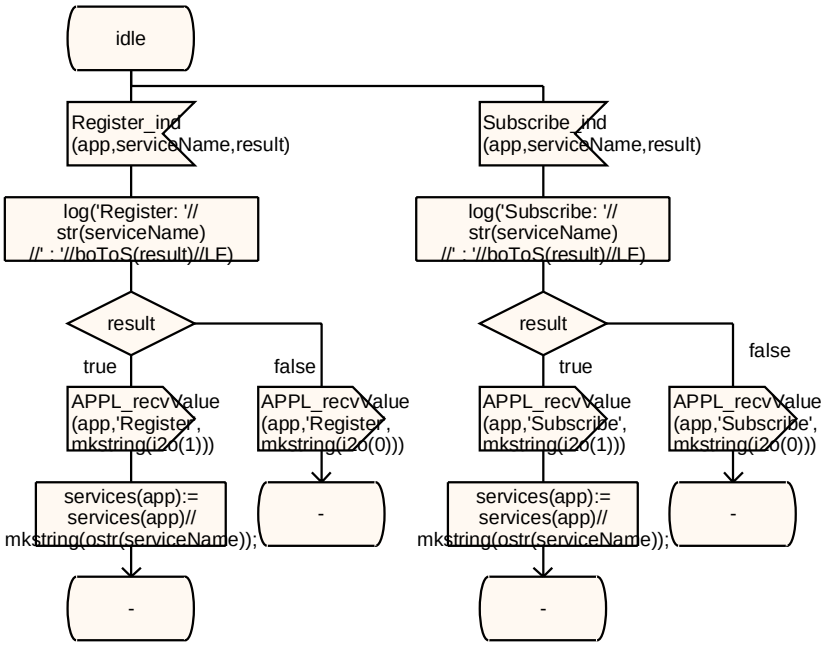
process CoDec 2(2)



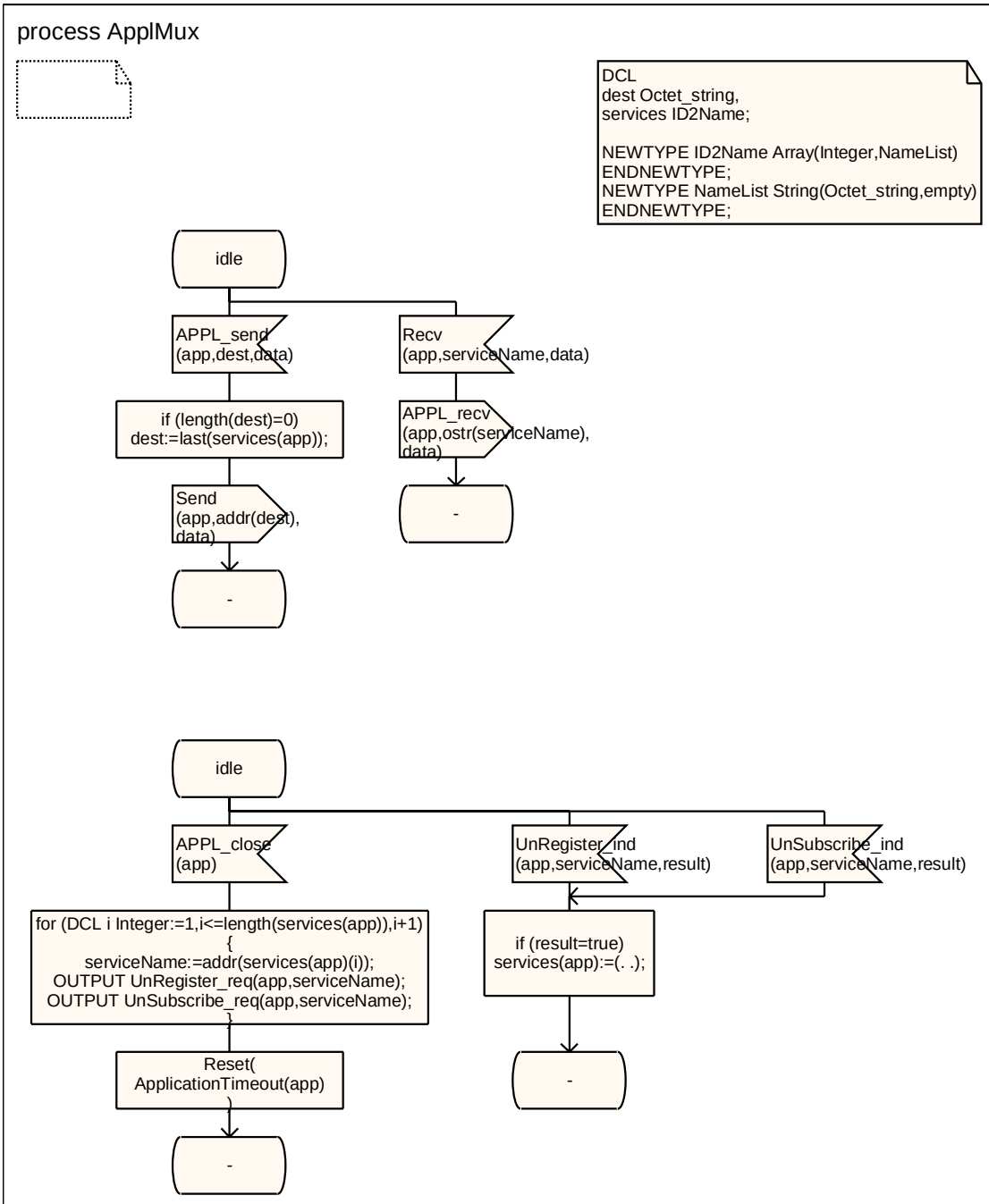
5.4 Application Interface



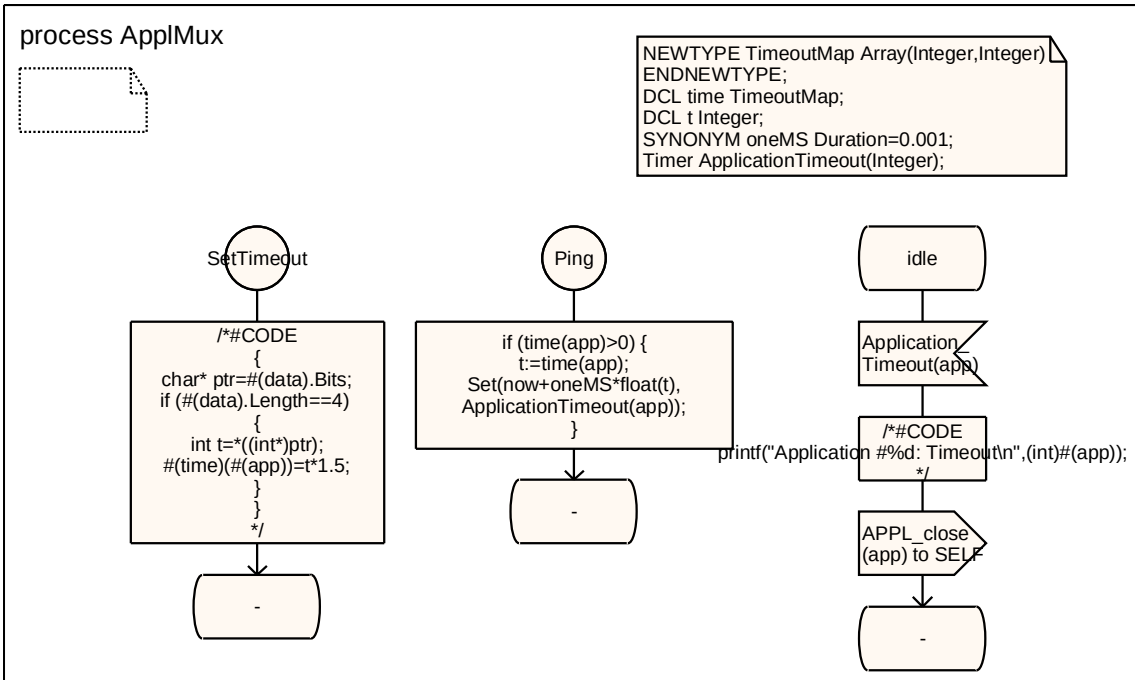



idle
Register_ind
(app,serviceName,result)
Subscribe_ind
(app,serviceName,result)
log('Register: '//
str(serviceName)
//' : '//boToS(result)//LF)
log('Subscribe: '//
str(serviceName)
//' : '//boToS(result)//LF)
result
result
true
false
true
false
APPL_recvValue
(app,'Register',
mkstring(i2o(1)))
APPL_recvValue
(app,'Register',
mkstring(i2o(0)))
APPL_recvValue
(app,'Subscribe',
mkstring(i2o(1)))
APPL_recvValue
(app,'Subscribe',
mkstring(i2o(0)))
services(app):=
services(app)//
mkstring(ostr(serviceName));
services(app):=
services(app)//
mkstring(ostr(serviceName));
-
-
-
-

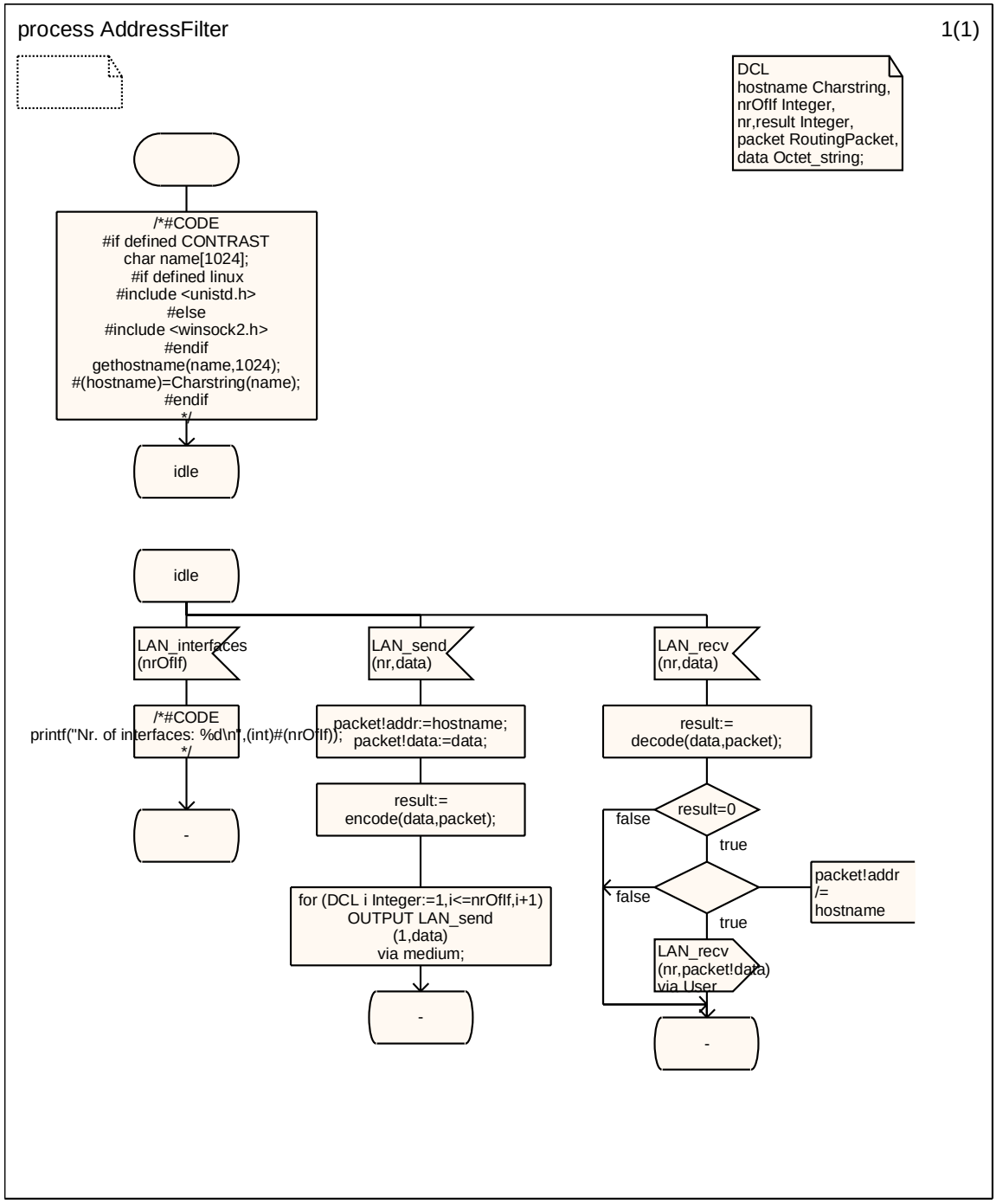
process ApplMux 3(4)



process ApplMux



5.5 Used Datatypes



package AddressType 1(1)

```
NEWTYPE AddressType inherits Charstring
OPERATORS ALL;
ADDING OPERATORS
matches: AddressType,AddressType -> Boolean;
addr: Octet_string -> AddressType;
str: AddressType -> Charstring;
ostr: AddressType -> Octet_string;


OPERATOR matches; FPAR a AddressType,b AddressType;RETURNS Boolean;
START;
return a=b;
ENDOPERATOR;

OPERATOR addr; FPAR os Octet_string;RETURNS AddressType;
DCL i Integer;
DCL s AddressType;
START;
task {
for (i:=1,i<=length(os),i+1)
   s:=s//mkstring(chr(o2i(os(i))));
};
return s;
ENDOPERATOR;

OPERATOR str; FPAR a AddressType;RETURNS Charstring;
DCL i Integer;
DCL s Charstring;
START;
task {
for (i:=1,i<=length(a),i+1)
   s:=s//mkstring(a(i));
};
return s;
ENDOPERATOR;

OPERATOR ostr; FPAR a AddressType;RETURNS Octet_string;
DCL i Integer;
DCL s Octet_string;
START;
task {
for (i:=1,i<=length(a),i+1)
   s:=s//mkstring(i2o(num(a((i)))));
};
return s;
ENDOPERATOR;


ENDNEWTYPE;
```