Telephone Exchange Interface
Design Documentation

Dominik Radziszowski
Bartosz Klimek
e-mail:{radzisz,bartekk}@student.uci.agh.edu.pl

University of Mining and Metallurgy
Cracow, Poland

June 1999

Contents

1 Introduction

This document describes the design of Telephone Exchange Interface (TEI). The primary function of this subsystem is to control the hardware (Dialogic boards) and switch incoming telephone calls to appropriate telephones, according to the number entered by the caller and a reply received from Call Redirection Manager, which is a remote server reachable through CORBA. More details about the general concept of the whole system and relations between its elements can be found in: "Call Redirection System", a document available at *http://iris.ics.agh.edu.pl/~bsklr*.

2 Functional Description

TEI's users are people who dial telephone numbers assigned to our system or pick up handsets of the telephones connected directly to the computer on which TEI is running ("internal telephones"). From their point of view, TEI is very simple. The following subsections show three services available from our system.

2.1 Service for an External User

When an external user ("caller") calls TEI, it answers the call and plays a welcome message waiting for digits from the user. He can enter either a physical number of an internal telephone or a logical number of a user registered in Call Redirection System (CRS). If the latter user ("receiver") is present and wants to answer phones, TEI switches the call to a telephone nearest to him. The users can talk. As soon as one of them hangs up, the call is terminated. It is possible that the system reacts to the incoming call in another way: by playing a message or generating a busy signal. It depends on how CRS is configurated and where the receiver is.

If the number is physical, the call is switched directly to the appropriate telephone.

Of course, if any non-critical error happens, the caller knows about it because a special message is played. If the caller hangs up before the call is switched, TEI stops handling it.

2.2 Service for an Internal User — an Internal Call

When an internal user (registered or not in CRS) picks up the handset of an internal telephone, TEI reacts by generating dial tone. The further actions are the same as above.

2.3 Service for an Internal User — an External Call

When an internal user (registered or not in CRS) picks up the handset of an internal telephone and press '0' as the first digit, he is switched directly to the real telephone exchange, so that he can dial any number. TEI just waits for any party to hang up, and terminates the call.

3 General Architecture

3.1 Hardware Dependencies

The design of Telephone Exchange Interface is strictly dependent on the architecture of the hardware. Two PC boards have been used: D41/ESC and MSI/SC. The former has four voice devices and four analog devices, to which external lines can be attached. The latter board has 24 analog devices, to which internal telephones can be attached. They are essentially independent from one another and multiple tasks can be in progress simultaneously. A natural solution to this is to use threads, one for each call. It is simple and quite effective. However, if there were more devices available, context switching overhead could be high.

3.2 Synchronization and Resource Management

A necessity of thread synchronization that arises here is obvious. Two or more threads may happen to compete to use one of the devices. Another, special thread ("main thread") is needed. It is responsible for resource management and controlling the other threads ("handlers"). It is usually asleep, waiting for an event from a device or a request from a handler.

Events meaningful for the main thread are: "ring" event from the voice device connected to the external line through its analog device and "off-hook" event from an MSI device. The former is triggered when an incoming call is present, the latter — when an internal user picks up the handset of an internal telephone. Reactions to those events are similar but not identical. The device from which the event came is removed from the list of free devices and a free handler is removed from the list of free handlers. Then the handler is woken up and ordered to handle the call. The difference is that there are two kinds of handlers: for MSI devices and for voice devices (further referred to as "D41 devices"), and they are kept in two separate lists of free handlers.

The requests to which the main thread replies are related to resource management: obtaining and freeing devices, and freeing handlers. When a handler wants, for example, to get a free device, it calls a method of the main thread, which wakes up the thread and waits for an answer (i.e. the handle of a free device or null value, meaning that no devices are available). From the handler's point of view, this is synchronous communication.

After the main thread finishes the task it was woken up for it continues to sleep waiting for events from free devices or requests from currently active handlers.

When a handler ends its job (e.g. the call terminates), it frees all resources it has reserved, frees itself (all that in cooperation with the main thread) and goes back to sleep.

4 Design Details

This sections depicts details of the structure of the system, collaboration of objects and specification of the algorithms.

4.1 Classes and Objects

Figure 1 presents the classes of TEI and links between them. Free_Resource_List is a list containing all devices currently not used (i.e. free). Free_D41_Handlers and Free_MSI_Handlers are pools of threads waiting for tasks. Each of these classes have only one instance — should be implemented as fields of Main_Thread class, which is a singleton, too.

Call_Switcher is a helper class, and there is one instance of it for each handler object. This class covers the logic of call switching. Because it depenends on the source of the incoming call (MSI device or D41 device), this class can be split into two separate classes during the implementation stage.

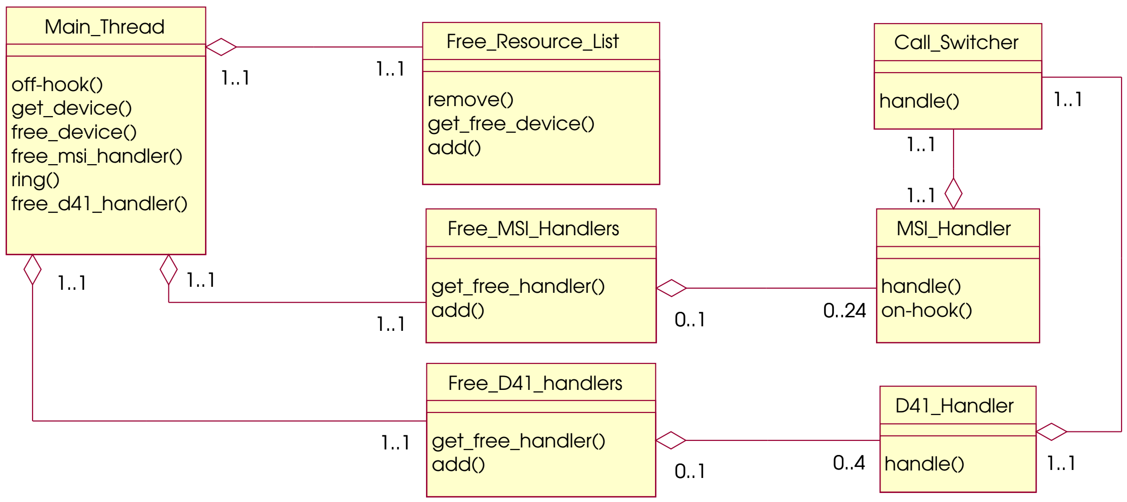

Figure 1: Classes of Telephone Exchange Interface

4.2 Behaviour of Objects

4.2.1 Scenarios

Particular scenarios are crucial for good understanding of the model of the system dynamics. Following this statement we will begin the description of the system behaviour with collaboration and sequence diagrams.

Figures 2–4 show how objects cooperate. The sequences of those operations are shown in Figures 5–7.

Many of the operations concern changing SCBus routing and it is recommended to get familiar with that subject before going deeper into the diagrams (see "SCBus Routing Guide" and other Dialogic manuals and documentation).

Let us briefly go over the diagrams. It is noteworthy that each collaboration and sequence diagram is based on some assumptions because the algorithms have a lot of branches.

- Figures 2 and 5 present the operations carried out by TEI when an inernal user picks the handset of an internal phone. We assume that the caller chooses a logical number, so the handler must ask CRM to decide what to do. The reply is to switch the call to another internal phone, possible nearest to the user with the requested logical number, and this is what is done. After some time, the caller hangs up and the call terminates.

- In figures 3 and 6 is the sequence of operations executed when an internal user picks up the handset of an internal telephone and wants to dial an external number. First he dials a '0' and then he gets connected with the external telephone interface. An interesting thing here is the fact that the D41 device originally assigned to the handler is not attached to an external line in this scenario. This is why it must be freed and the handler reserves another D41 device, with an external line.

- Figures 4 and 7 show what happens when an external user calls our system. This is very similar to the first scenario — from the logical point of view. They differ in technical details because different devices are engaged here.

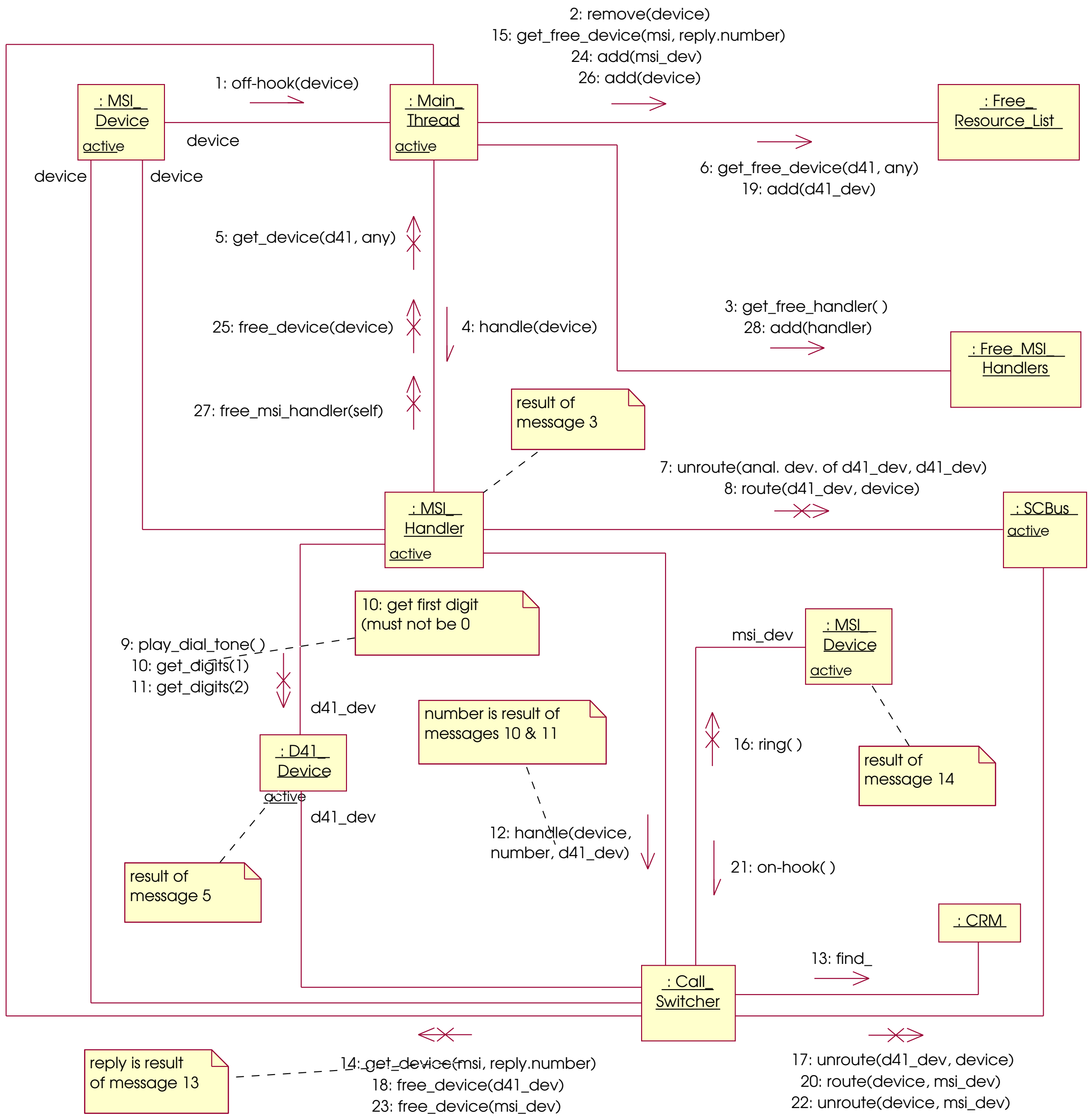

Figure 2: A call from an internal telephone to an internal user given by his logical number

4.2.2 Algorithms

The behaviour of active objects is described with state diagrams and operation specifications (Appendix A). The actions found in the diagrams are written using pseudosyntax similar to Pascal. "See documentation" means that the action is more complex and its specification can be found in Appendix A.

State diagrams have been developed for the following classes:

- Main_Thread (Figure 8)
- MSI_Handler (Figure 9)
- D41_Handler (Figure 10)
- Call_Switcher (Figure 11)

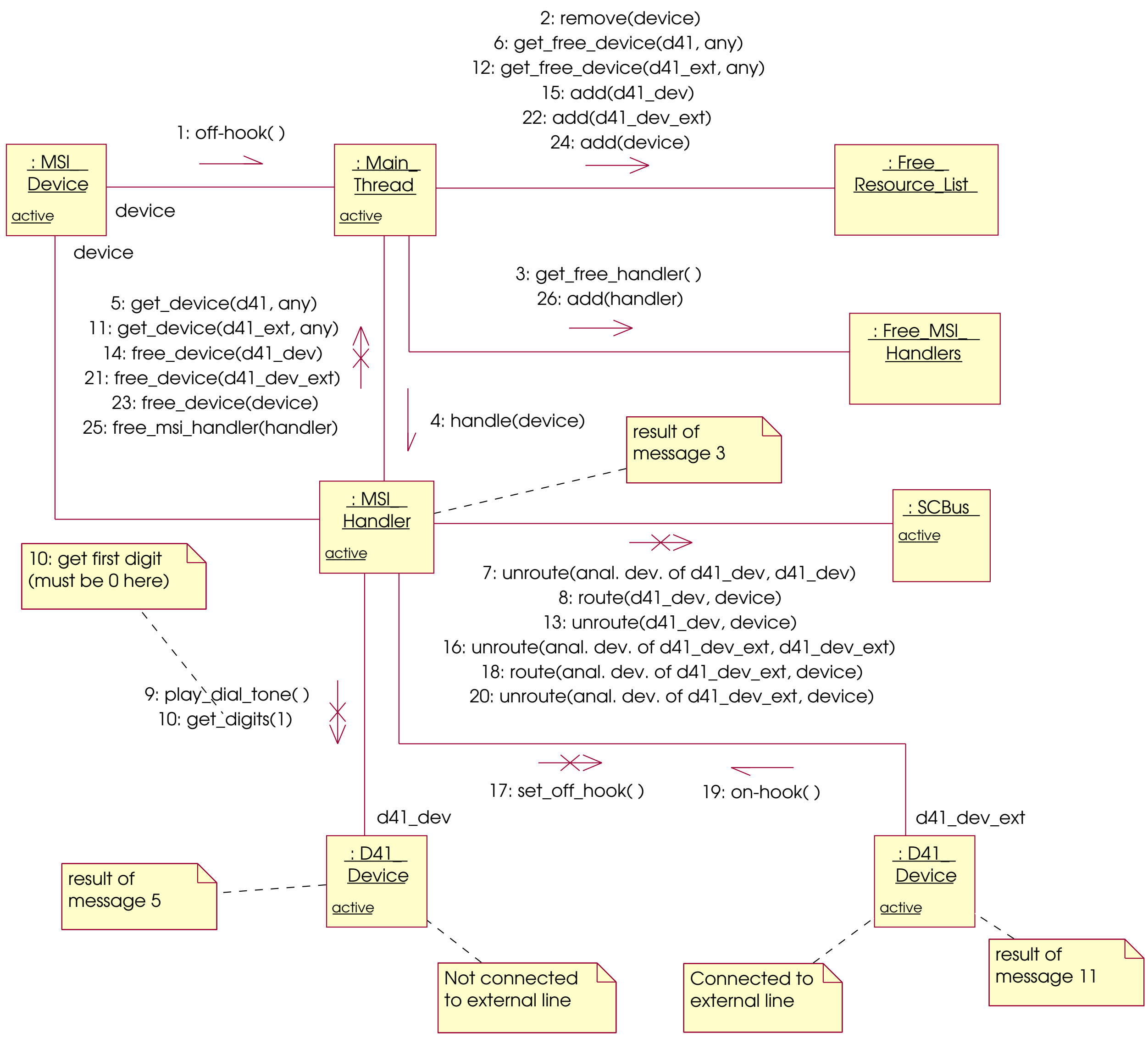

Figure 3: A call from an internal telephone to an external number

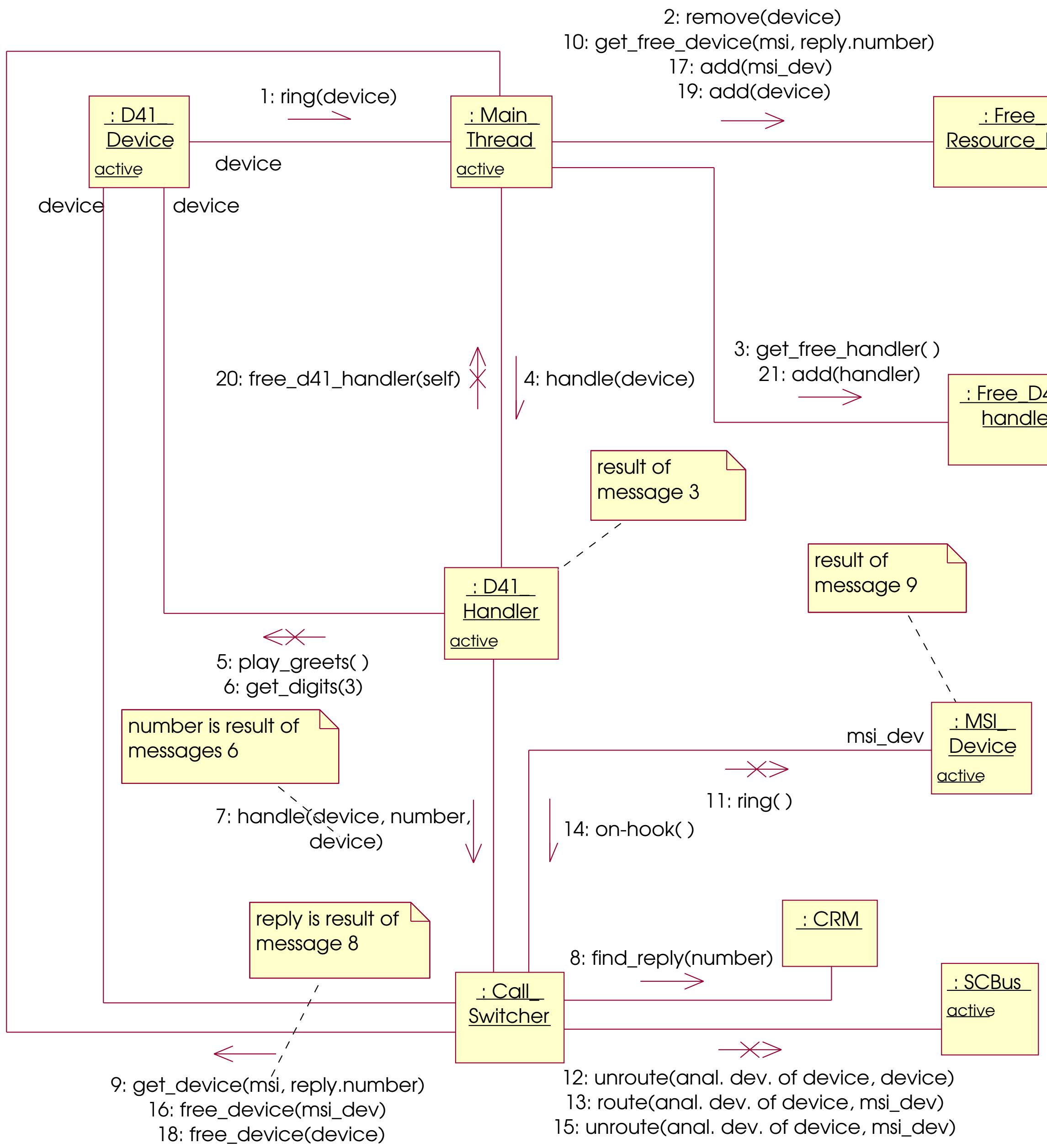

Figure 4: A call from an external number to an internal user given by his logical number

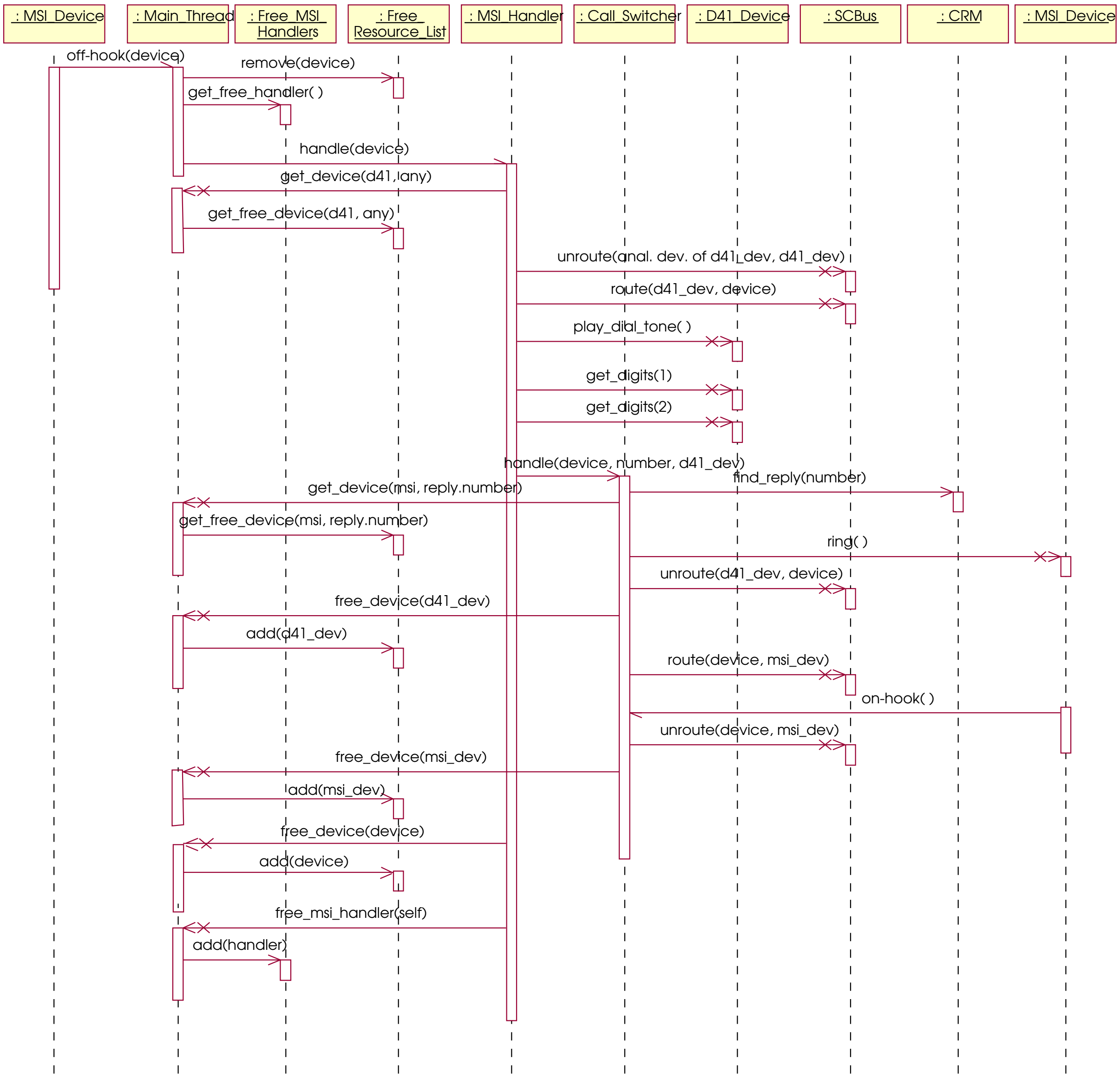

Figure 5: A call from an internal telephone to an internal user given by his logical number

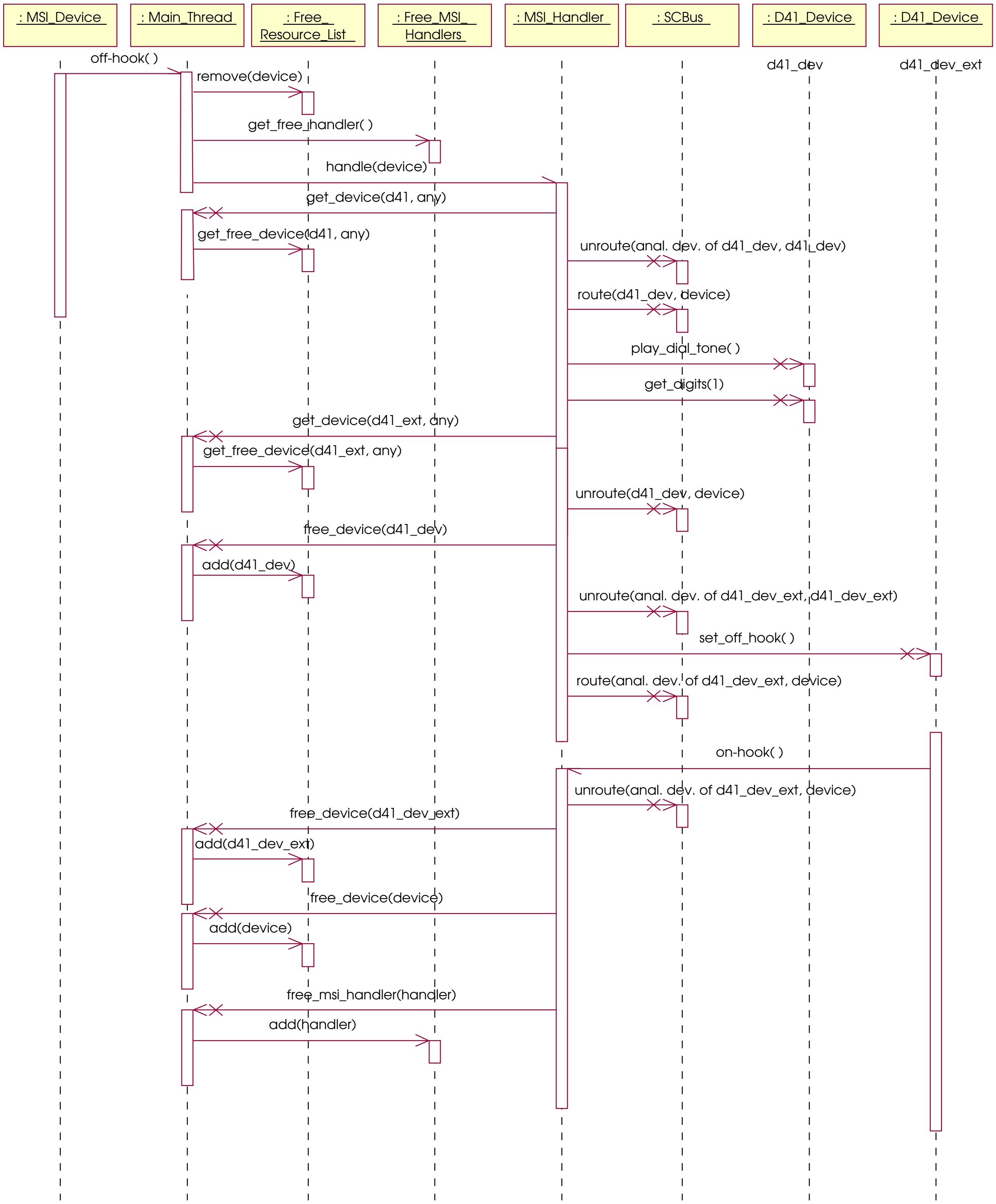

Figure 6: A call from an internal telephone to an external number

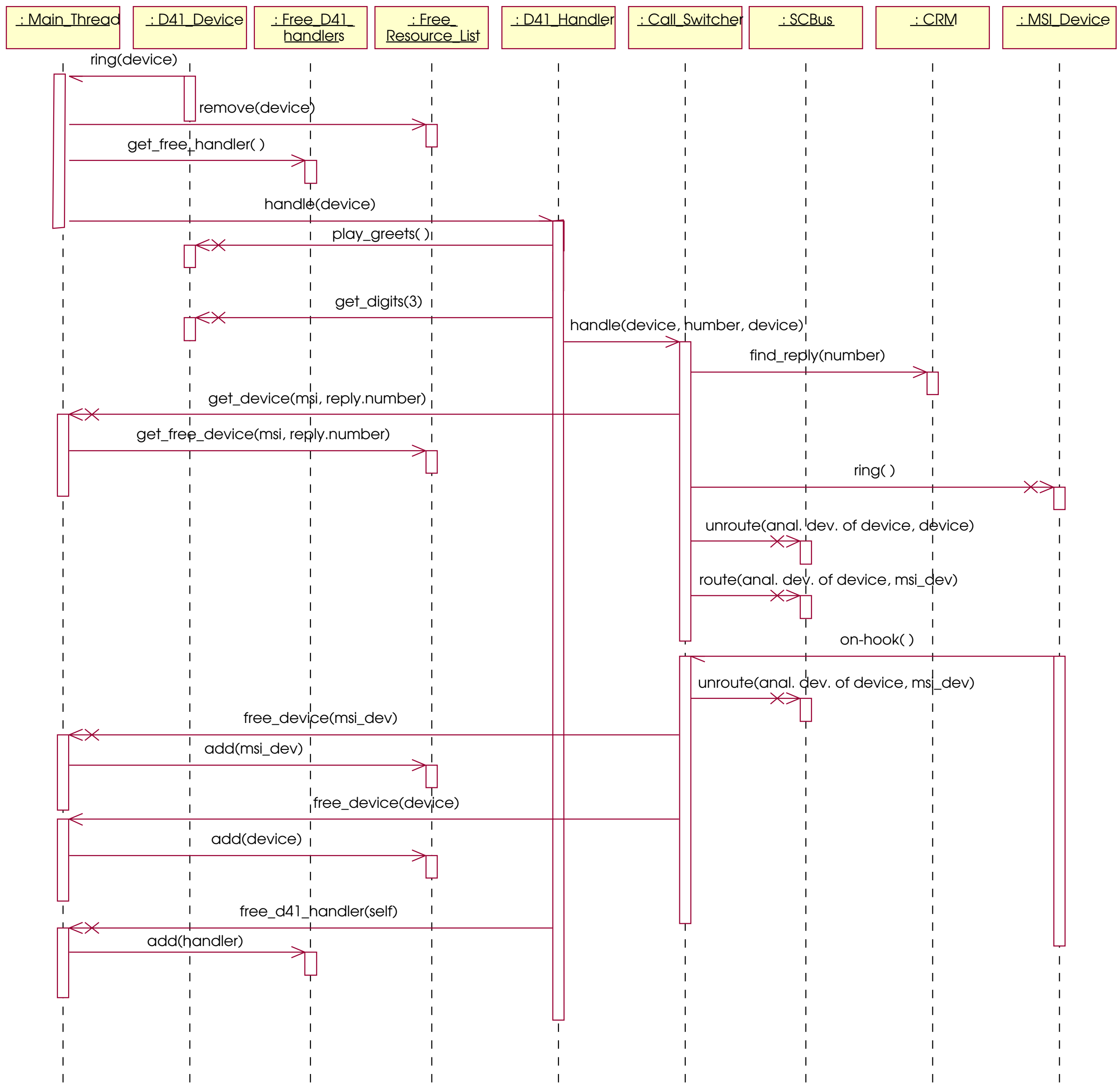

Figure 7: A call from an external number to an internal user given by his logical number

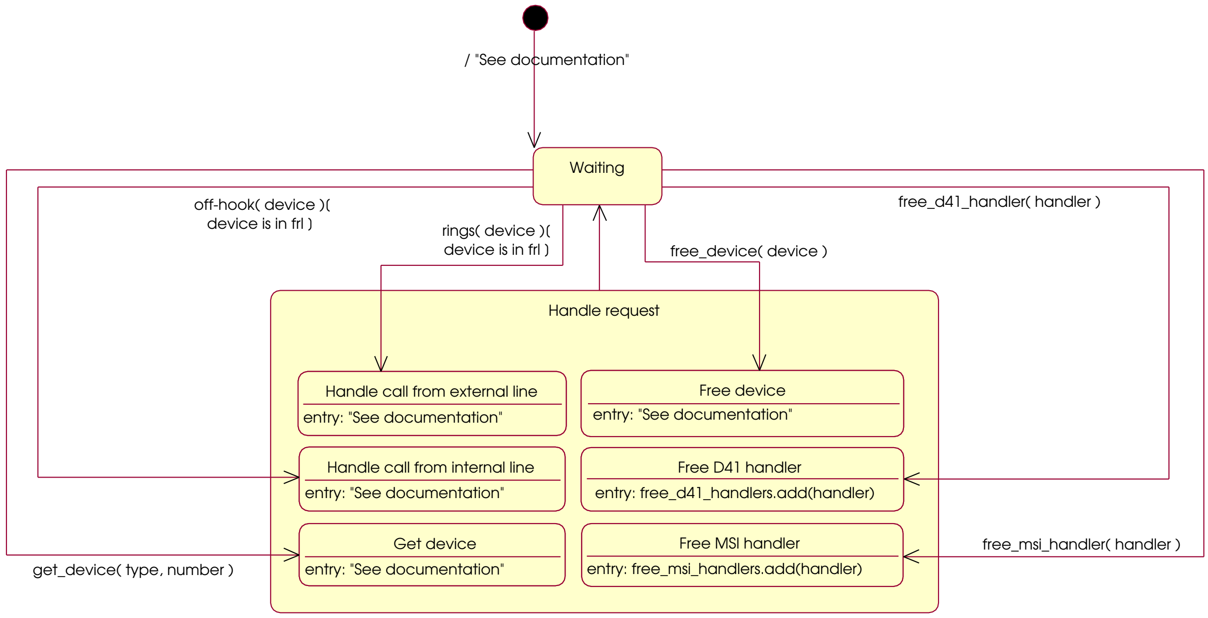

Figure 8: State Diagram for Main_Thread

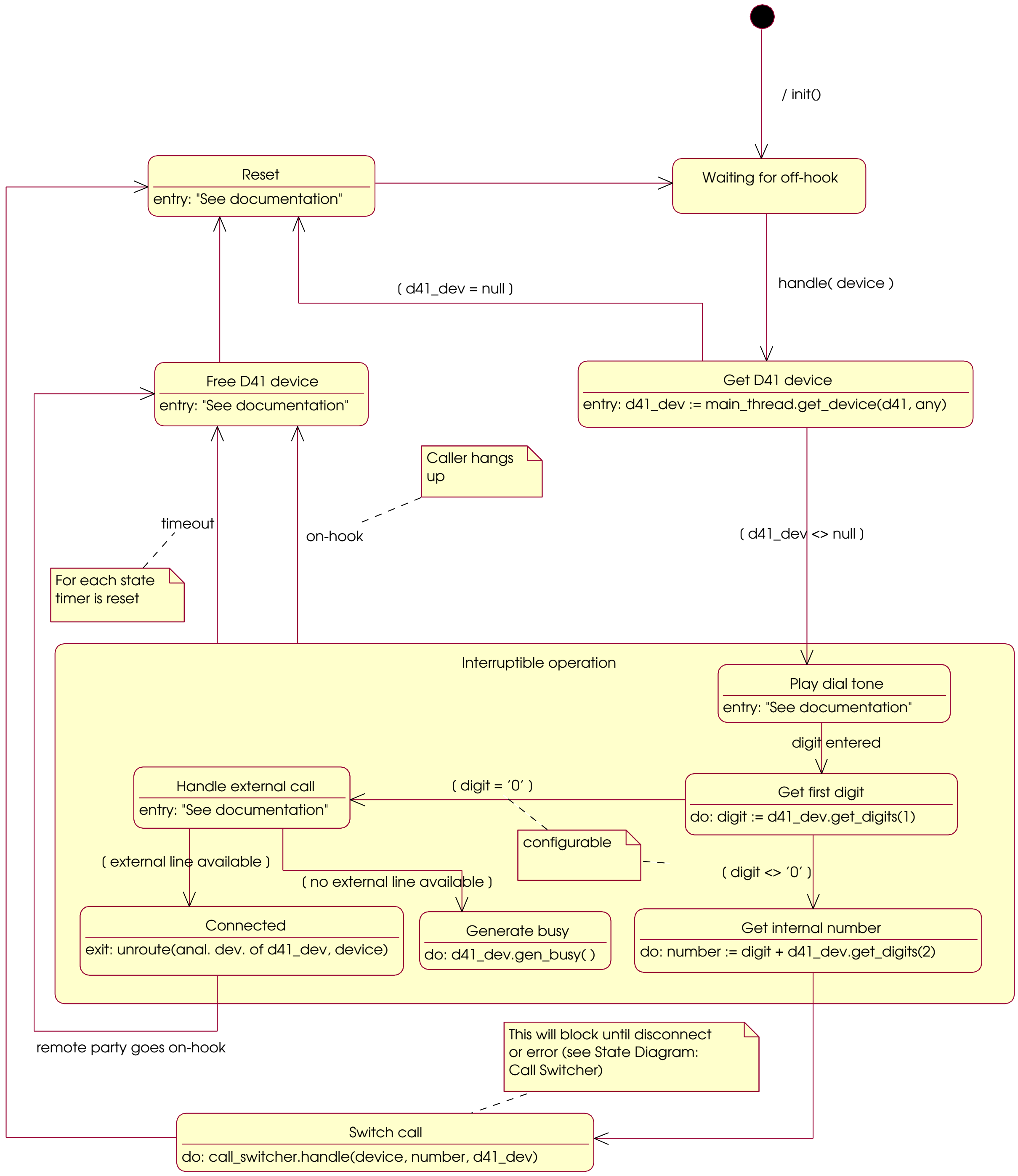

Figure 9: State Diagram for MSI_Handler

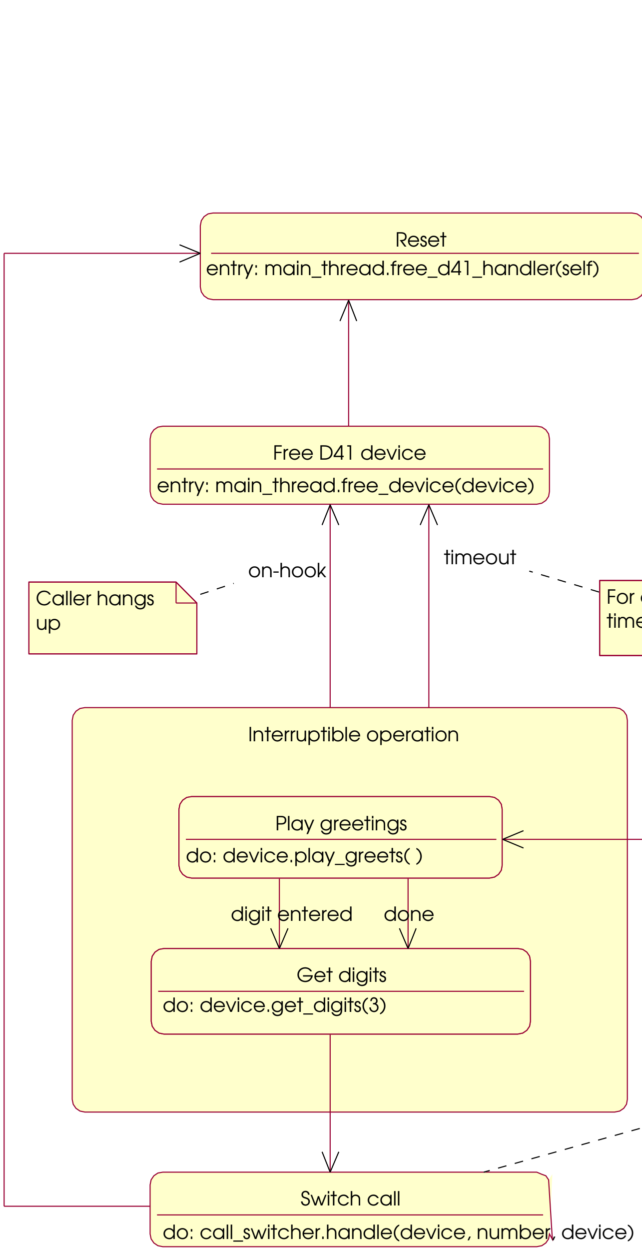

Figure 10: State Diagram for D41_Handler

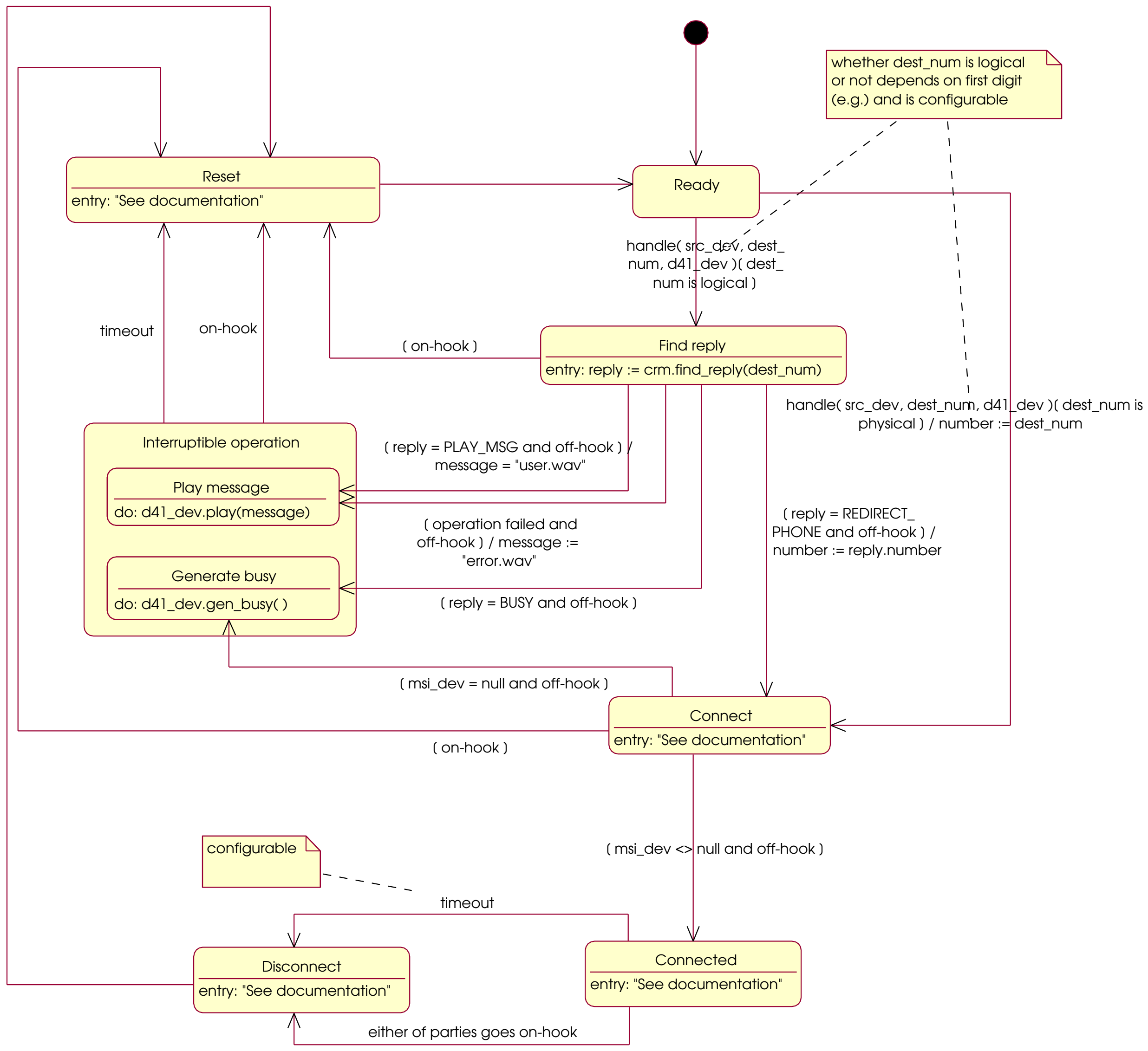

Figure 11: State Diagram for Call_Switcher

A Specification

Class name:

Main_Thread

Category: Logical View
External Documents:
Export Control: Public
Cardinality: 1..1
Hierarchy:
Superclasses: none
Associations:

<no rolename> : Free_Resource_List in association <unnamed>
<no rolename> : Free_D41_handlers in association <unnamed>
<no rolename> : Free_MSI_Handlers in association <unnamed>

Public Interface:
Operations:

off-hook
get_device
free_device
free_msi_handler
ring
free_d41_handler

State machine: Yes
Concurrency: Active
Persistence: Transient

State name:

Waiting

Type: Normal

State name:

Type: StartState

State name:

Handle request

Type: Normal
SubStates:
Handle call from external line, Handle call from internal line, Get device, Free device, Free MSI handler, Free D41 handler

History: Off

State name:

Handle call from external line

Type: Normal
Parent State: Handle request
Documentation:

```
entry/
device.set_off_hook( );
frl.remove(device);
handler := free_d41_handlers.get_free_handler( );
handler.handle(device)
```

Actions:
entry: "See documentation"

History: Off

State name:

Handle call from internal line

Type: Normal
Parent State: Handle request
Documentation:

```
entry/
frl.remove(device);
handler := free_msi_handlers.get_free_handler( );
handler.handle(device)
```

Actions:
entry: "See documentation"

History: Off

State name:

Get device

Type: Normal
Parent State: Handle request
Documentation:

```
entry/
device := frl.get_free_device(type, number);
return device
```

Actions:
entry: "See documentation"

History: Off

State name:

Free device

Type: Normal
Parent State: Handle request
Documentation:

entry/
if device type is MSI then
route(device, device);
else
route(device, analog device of device);
device.set_on_hook( );
end if;
frl.add(device)

Actions:

entry: "See documentation"

History: Off

State name:

Free MSI handler

Type: Normal
Parent State: Handle request
Actions:

entry: free_msi_handlers.add(handler)

History: Off

State name:

Free D41 handler

Type: Normal
Parent State: Handle request
Documentation:

Actions:

entry: free_d41_handlers.add(handler)

History: Off

Transition from <u>Waiting</u> *to* <u>Handle call from external line</u>

rings(device)(device is in frl)

Transition from <u>Waiting</u> *to* <u>Handle call from internal line</u>

off-hook(device)(device is in frl)

Transition from Waiting *to* Get device
get_device(type, number)

Transition from Waiting *to* Free device
free_device(device)

Transition from Waiting *to* Free MSI handler
free_msi_handler(handler)

Transition from Waiting *to* Free D41 handler
free_d41_handler(handler)

Transition from StartState *to* Waiting
/ "See documentation"

Documentation:
At startup:
for each msi device do
route it to itself;
end for;
for each d41 device do
route it to its analog device;
set it on-hook;
end for
add all devices to frl;
create 24 msi handlers, add them to free_msi_handlers;
create d41 handlers (as many as many external lines we have), add them to free_d41_handlers

Transition from Handle request *to* Waiting

Class name:

Call_Switcher

Category: Logical View
External Documents:
Export Control: Public
Cardinality: n
Hierarchy:
Superclasses: none
Associations:

<no rolename> : D41_Handler in association <unnamed>
<no rolename> : MSI_Handler in association <unnamed>

Public Interface:
Operations:
handle

State machine: Yes
Concurrency: Sequential
Persistence: Transient

State name:

Type: StartState

State name:

Ready

Type: Normal

State name:

Find reply

Type: Normal
Documentation:

Actions:
entry: reply := crm.find_reply(dest_num)

State name:

Reset

Type: Normal
Documentation:

```
entry/
if d41_dev <> null then
if src_dev <> d41_dev then
unroute(d41_dev, src_dev);
end if;
main_thread.free_device(d41_dev);
d41_dev := null;
end if;
if msi_dev <> null then
main_thread.free_device(msi_dev);
msi_dev := null;
end if;
```

Actions:
entry: "See documentation"

State name:
Interruptible operation

Type: Normal
SubStates:
Generate busy, Play message

History: Off

State name:
Connected

Type: Normal
Documentation:
entry/
if d41_dev <> src_dev then
unroute(d41_dev, src_dev);
main_thread.free_device(d41_dev);
d41_dev := null;
route(src_dev, msi_dev);
else
route(analog device of d41_dev, msi_dev);
end if;

Actions:
entry: "See documentation"

State name:
Disconnect

Type: Normal
Documentation:
entry/
if src_dev <> d41_dev then
unroute(src_dev, msi_dev);
else
unroute(analog device of d41_dev, msi_dev);
end if

Actions:
entry: "See documentation"

State name:

Connect

Type: Normal

Documentation:

```
entry/
msi_dev := main_thread.get_device(msi, number);
if msi_dev <> null and caller#s off-hook then
msi_dev.ring();
if no answer or busy then
main_thread.free_device(msi_dev);
msi_dev := null;
end if;
end if
```

Actions:

entry: "See documentation"

State name:

Generate busy

Type: Normal

Parent State: Interruptible operation

Actions:

do: d41_dev.gen_busy()

History: Off

State name:

Play message

Type: Normal

Parent State: Interruptible operation

Actions:

do: d41_dev.play(message)

History: Off

Transition from StartState *to* Ready

Transition from Ready *to* Find reply

handle(src_dev, dest_num, d41_dev)(dest_num is logical)

Transition from Ready *to* Connect

handle(src_dev, dest_num, d41_dev)(dest_num is physical) / number := dest_num

Transition from Find reply *to* Reset
(on-hook)

Transition from Find reply *to* Play message
(operation failed and off-hook) / message := "error.wav"

Transition from Find reply *to* Connect
(reply = REDIRECT_PHONE and off-hook) / number := reply.number

Transition from Find reply *to* Play message
(reply = PLAY_MSG and off-hook) / message = "user.wav"

Transition from Find reply *to* Generate busy
(reply = BUSY and off-hook)

Transition from Reset *to* Ready

Transition from Interruptible operation *to* Reset
on-hook

Transition from Interruptible operation *to* Reset
timeout

Transition from Connected *to* Disconnect
either of parties goes on-hook

Transition from Connected *to* Disconnect timeout

Transition from Disconnect *to* Reset

Transition from Connect *to* Connected
(msi_dev <> null and off-hook)

Transition from Connect *to* Reset
(on-hook)

Transition from Connect *to* Generate busy
(msi_dev = null and off-hook)

Class name:

MSI_Handler

Category: Logical View
External Documents:
Export Control: Public
Cardinality: n
Hierarchy:
Superclasses: none
Associations:

<no rolename> : Free_MSI_Handlers in association <unnamed>
<no rolename> : Call_Switcher in association <unnamed>

Public Interface:
Operations:
handle
on-hook

State machine: Yes
Concurrency: Active
Persistence: Transient

State name:

Type: StartState

State name:

Waiting for off-hook

Type: Normal

State name:

Get D41 device

Type: Normal

Actions:

entry: d41_dev := main_thread.get_device(d41, any)

State name:

Reset

Type: Normal

Documentation:

entry/
main_thread.free_device(device);
main_thread.free_msi_handler(self)

Actions:

entry: "See documentation"

State name:

Free D41 device

Type: Normal

Documentation:

entry/
unroute(d41_dev, device);
main_thread.free_device(d41_dev)

Actions:

entry: "See documentation"

State name:

Interruptible operation

Type: Normal

SubStates:

Play dial tone, Get first digit, Get internal number, Handle external call, Generate busy, Connected

History: Off

State name:

Switch call

Type: Normal
Actions:
do: call_switcher.handle(device, number, d41_dev)

State name:

Play dial tone

Type: Normal
Parent State: Interruptible operation
Documentation:
entry/
unroute(d41_dev, analog device of d41_dev);
route(d41_dev, device)
do/
d41_dev.play_tone( )

Actions:
entry: "See documentation"

History: Off

State name:

Get first digit

Type: Normal
Parent State: Interruptible operation
Actions:
do: digit := d41_dev.get_digits(1)

History: Off

State name:

Get internal number

Type: Normal
Parent State: Interruptible operation
Actions:
do: number := digit + d41_dev.get_digits(2)

History: Off

State name:

Handle external call

Type: Normal
Parent State: Interruptible operation
Documentation:

entry/
external line available := true;
if d41_dev is not connected to external line then
dev_ext := main_thread.get_device(d41_ext, any);
if dev_ext <> null then
unroute(d41_dev, device);
main_thread.free_device(d41_dev);
d41_dev := dev_ext;
unroute(d41_dev, analog device of d41_dev);
else
external line available := false;
end if;
else
unroute(d41_dev, device);
end if;
if external line available then
d41_dev.set_off_hook( );
route(analog device of d41_dev, device);
end if

Actions:

entry: "See documentation"

History: Off

State name:

Generate busy

Type: Normal
Parent State: Interruptible operation
Actions:

do: d41_dev.gen_busy( )

History: Off

State name:

Connected

Type: Normal
Parent State: Interruptible operation
Actions:

exit: unroute(anal. dev. of d41_dev, device)

History: Off

Transition from StartState *to* Waiting for off-hook

/ init()

Transition from Waiting for off-hook *to* Get D41 device
handle(device)

Transition from Get D41 device *to* Reset
(d41_dev = null)

Transition from Get D41 device *to* Play dial tone
(d41_dev <> null)

Transition from Reset *to* Waiting for off-hook

Transition from Free D41 device *to* Reset

Transition from Interruptible operation *to* Free D41 device
timeout

Transition from Interruptible operation *to* Free D41 device
on-hook

Transition from Switch call *to* Reset

Transition from Play dial tone *to* Get first digit
digit entered

Transition from Get first digit *to* Get internal number
(digit <> #0#)

Transition from Get first digit *to* Handle external call
(digit = #0#)

Transition from <u>Get internal number</u> *to* <u>Switch call</u>

Transition from <u>Handle external call</u> *to* <u>Generate busy</u>
(no external line available)

Transition from <u>Handle external call</u> *to* <u>Connected</u>
(external line available)

Transition from <u>Connected</u>*to* <u>Free D41 device</u>
remote party goes on-hook

Class name:

D41_Handler

Category: Logical View
External Documents:
Export Control: Public
Cardinality: n
Hierarchy:
Superclasses: none
Associations:

<no rolename> : Free_D41_handlers in association <unnamed>
<no rolename> : Call_Switcher in association <unnamed>

Public Interface:
Operations:
handle

State machine: Yes
Concurrency: Active
Persistence: Transient

State name:

Type: StartState

State name:

Waiting for ring

Type: Normal

State name:

Reset

Type: Normal
Documentation:

Actions:
entry: main_thread.free_d41_handler(self)

State name:

Interruptible operation

Type: Normal
SubStates:
Play greetings, Get digits

History: Off

State name:

Free D41 device

Type: Normal
Actions:
entry: main_thread.free_device(device)

State name:

Switch call

Type: Normal
Actions:
do: digit := device.get_digits(3)

State name:

Play greetings

Type: Normal
Parent State: Interruptible operation
Actions:
do: device.play_greets( )

History: Off

State name:

Get digits

Type: Normal
Parent State: Interruptible operation
Actions:
do: digit := device.get_digits(3)

History: Off

Transition from StartState *to* Waiting for ring
/ init()

Transition from Waiting for ring *to* Play greetings
handle(device)

Transition from Reset *to* Waiting for ring

Transition from Interruptible operation *to* Free D41 device
timeout

Transition from Interruptible operation *to* Free D41 device
on-hook

Transition from Free D41 device *to* Reset

Transition from Switch call *to* Reset

Transition from Play greetings *to* Get digits
digit entered

Transition from Play greetings *to* Get digits
done

Transition from Get digits *to* Switch call