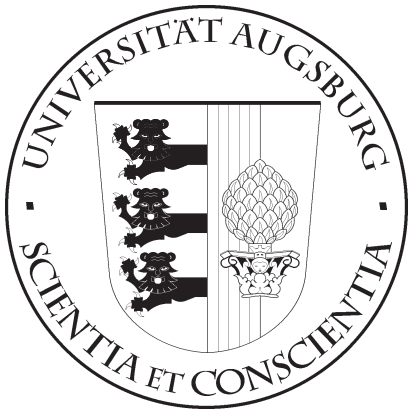
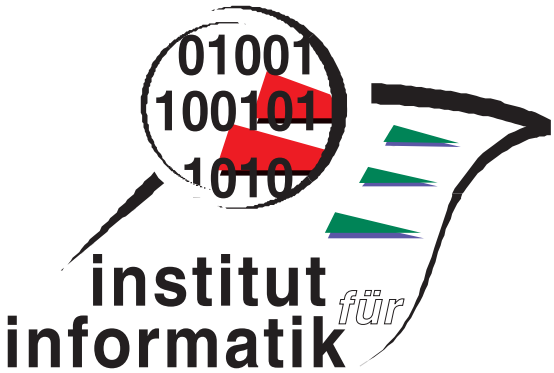
Universität Augsburg



electronic Ticketing — a Case-Study

Dominik Haneberg

Report 2001-9 Dezember 2001



Institut für Informatik
D-86135 Augsburg

Dominik Haneberg

electronic ticketing — a Case-Study

Abstract

The electronic ticketing was developed within the scope of the Go!Card project. It is used as a test object for the techniques for the development of secure smartcard applications that are developed in the Go!Card project.

The electronic ticketing case study deals with an e-commerce scenario for the electronic sale of railway or flight tickets. The customers buy their tickets from a server that transmits the signed and encrypted tickets to the customer, where they are loaded on the customers smartcard. Then the smartcard decrypts and verifies the tickets and stores them. The tickets are checked and obliterated offline by the train's conductor using a portable computer. This report describes the scenario of the electronic ticketing case study and explains some of the functions of the card program.

Contents

List of Figures

Chapter 1

The electronic Ticketing case study

To trial our new development method and the verification calculus we test our results on case studies. The central case study of the Go!Card project is the electronic Ticketing application. This is a e-commerce scenario dealing with the sales of railway our flight tickets. The central concept of electronic Ticketing is that the tickets no longer exist in a paper form. The tickets are ordered, issued, stored and checked solely electronic. The customer orders the tickets via a web interface on any PC with smartcard reader or via specialized terminals, which could be set up in railway stations or travel agencies. The order information is transmitted to the server of the issuer, which generates matching tickets and reservations and transmits them over the internet to the smartcard of the customer. The purchased tickets get stored on the smartcard and can be checked offline by the conductor if required. Therefore he uses a portable card reader that reads the ticket data from the smartcard and displays them. Using the same device the conductor obliterates used tickets. Cryptographic methods are used to prevent manipulation and to ensure that only valid tickets get stored on the smartcard.

1.0.1 Scenario

An overall picture of the electronic Ticketing scenario can be seen in figure 1.1. To realize the core services three central components are used. These are:

- The customer's smartcard
- The server of the ticket issuer
- The conductor's checking device

The smartcard contains the card application that handles the encrypted communication with the server of the ticket issuer, stores the tickets and shows and obliterates tickets in interaction with a conductor device. To be able to fulfill its tasks the smartcard must be able to perform cryptographic operations. The use of encryption ensures the confidentiality of the customer data, the use of digital signatures ensures integrity and authenticity of the transferred data. The server of the ticket issuer is activated by a customer who wants to buy a ticket. The server contacts the customer's smartcard and prepares for the transmission of a new ticket. Then he accepts the customer's order and generates a corresponding ticket, lets the user pay the ticket and transfers ticket data to the customer's smartcard.

The conductor's checking device is used in trains to check and obliterate the tickets of the passengers. To do so the conductor's device authenticates itself towards the smartcard,

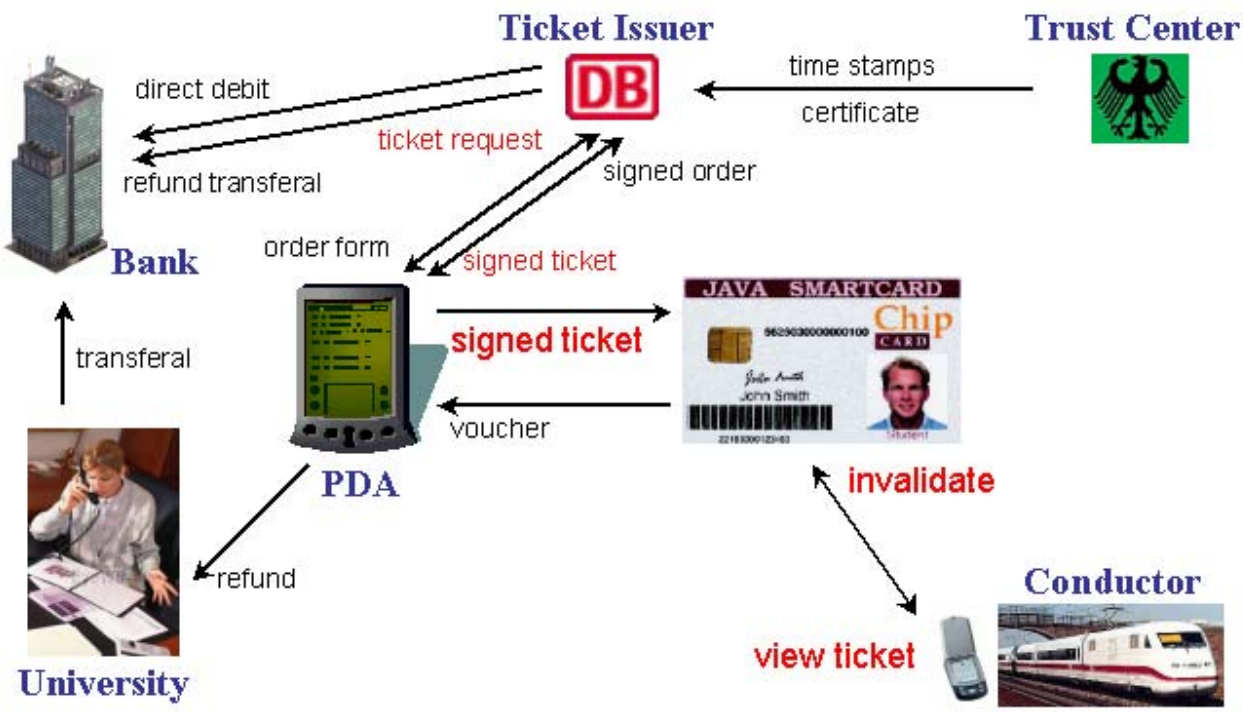


Figure 1.1: Structure of the electronic Ticketing case study

requests the ticket data and shows them to the conductor for verification. If the ticket is okay the checking device orders the smartcard to mark the ticket as obliterated. Thereby it is crucial that the verification of a ticket can be done offline, thus no network connection between the train (conductor) and ticket sales system is needed.

The cardlet also provides the possibility to pass on a ticket from one smartcard to another. Thereby it is important that the ticket is at no time the ticket is in usable condition on both cards simultaneously because this would permit to defraud the ticket issuer. The impossibility of imposture is ensured by using an appropriate communication protocol.

Additionally the application allows it to return unused tickets electronically. When an unused ticket should be returned to the issuer the cardlet transfers the ticket to the server which initiates the refund of the customer. To prevent fraud this function also requires appropriate protocols.

1.0.2 Extensions

The core services of the electronic Ticketing case study can be extended by further functions, for example the automatic refund of tickets bought for business trips.

Chapter 2

The protocols

2.1 Overview

The protocols are all specified as UML[RJB98, OMG99] activity diagrams. Each participant of a protocol is represented by a swimlane. Most often there are two participants, one terminal and one smartcard. The activity diagram shows the control flow and the communication of its protocol. The data exchange is represented by `command`- and `response`-objects. The `command`-objects are generated by the terminal and sent to the card, while `response`-objects are generated by the smartcard. We link a note to each object that contains information about the composition of a `command` or `response`. The activities are used to describe changes of the internal state of the terminal and the card. We use branches to express checks of preconditions for protocol steps.

The protocols all have a similar control flow. A protocol run is always initiated by the terminal. After being triggered off by the user, the terminal calls a function of the cardlet. Thereby the terminal can transmit data to the cardlet if necessary. The cardlet performs the requested function and returns data if required. It is not possible for the cardlet to start a protocol run of its own. Wrong data or unsatisfied security preconditions lead to an exception and the termination of the protocol run.

2.2 Add Check

2.2.1 Purpose

This protocol is used to add a check to a ticket on the card. The protocol is used by the conductor's portable card reader to obliterate the ticket for the current trip.

2.2.2 Preconditions

The card must be authenticated, this is done using the *Authenticate Cardlet*-protocol (Section 2.3), and the conductor terminal must have been authenticated using the *Authenticate Conductor*-protocol (Section 2.4). The ticketnumber which is transferred to the card must be a correct ticketnumber, i. e. the transmitted number must be less or equal to the number of tickets currently stored on the card. There mustn't be any other protocol running on the card, i. e. the `nxt_cmd`-field of the cardlet must be `any`.

2.2.3 The protocol

In the first step of the protocol the server tests if there is an internal error. A protocol run is only started if there's no error in the server. If there's no error the server checks of the cardlet was previously authenticated. If this is not the case the execution of the protocol is aborted. Otherwise the server stores the command that will be sent to the card (`last_cmd`) and sends a command to the card consisting of a card instruction (`CMD_addCheck`), the number of the ticket to which the check is to be added (`ticket_no`) and the check (`check`). The first thing the cardlet does after receiving this command is to check if there's cuurently another protocol running. This is done by checking if `nxt_cmd` is not equal `any`. If this condition isn't satisfied the cardlet generates a `SW_CONDITIONS_NOT_SATISFIED`-exception. If the test is successful the cardlet checks if the Conductor Terminal is properly authenticated. If the terminal isn't authenticated the cardlet produces a `SW_SECURITY_STATUS_NOT_SATISFIED`-exception. Otherwise the cardlet tests if the next precondition is satisfied. This precondtion ensures that the ticket number sent by the terminal (`ticket_no`) refer to a ticket that actually exists. If the number is inappropriate the cardlet produces a `SW_RECORD_NOT_FOUND`-exception. If there's a matching ticket, the cardlet stores the new check and informs the terminal of the successful completion of the operation by transmitting `SW_OK`. After receiving the expected response the terminal resets the `last_cmd`-field.

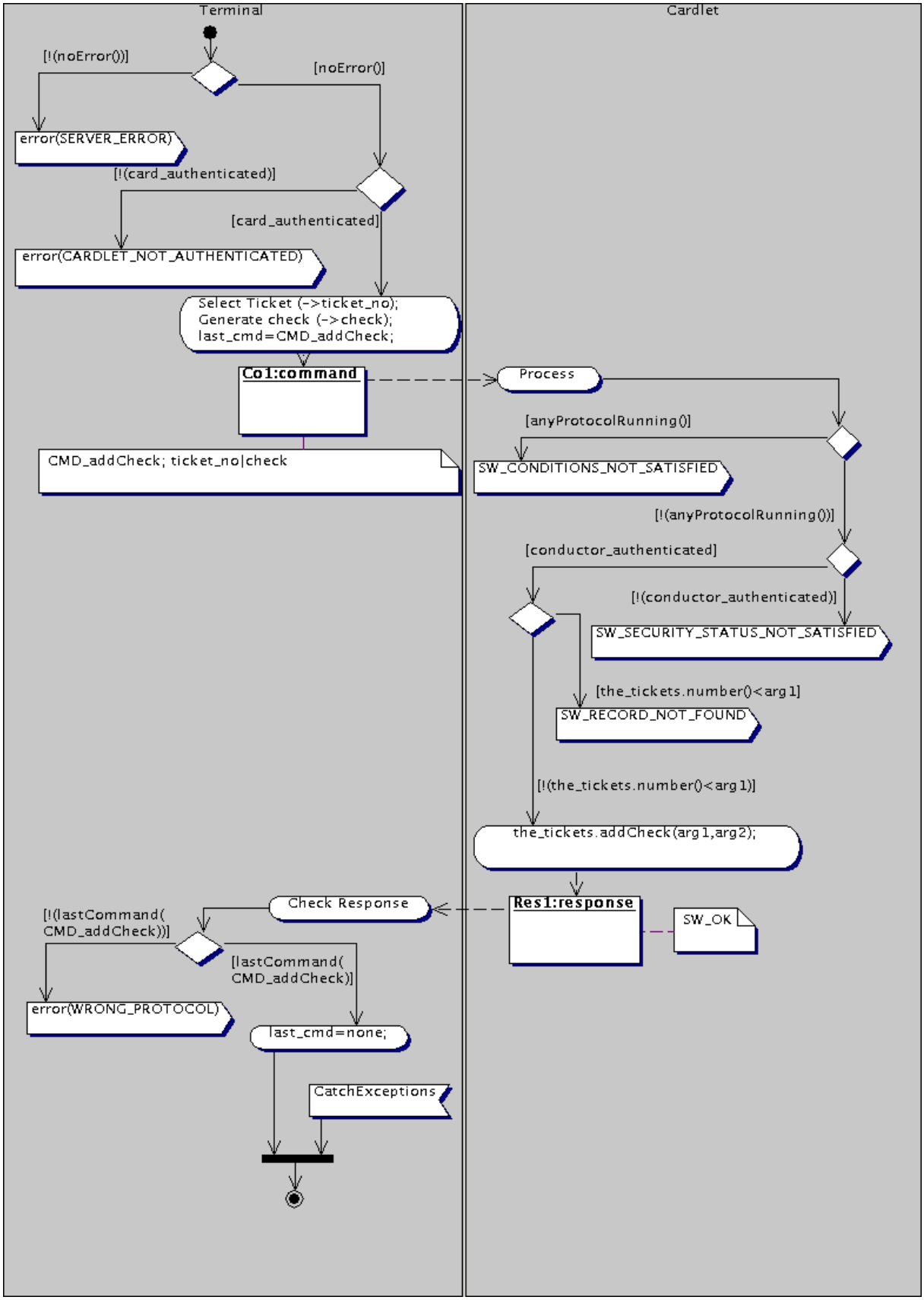


Figure 2.1: *Add Check*-protocol

2.3 Authenticate Cardlet

2.3.1 Purpose

This protocol is used to check the authenticity of a cardlet. Using this protocol a terminal can check if a cardlet is geniune or a fake. This protocol is used by the conductor terminal to check the authenticity of a cardlet to be sure that the tickets on the cardlet are to be considered as authentic.

2.3.2 Preconditions

Before this protocol can be executed the terminal must have acquired and verified the public key of the card. There mustn't be any other protocol running on the card, i. e. the `nxt_cmd`-field of the cardlet must be `any` and the user must be authenticated, i. e. the correct user PIN must have been entered using the *Authenticate User*-protocol (Section 2.5).

2.3.3 The protocol

As in all other protocols the first step in the *Authenticate Cardlet*-protocol is to test, if the terminal isn't in an error condition. If there's no error the terminal can advance to the next step, to check that the card key and the card key signature were retrieved from the card and that the signature is a valid certificate for the key. The protocol run will only continue if the signature is ok. If these tests were successfully passed, the terminal generates a challenge (a random value of fixed size), stores the challenge in the `terminal_challenge`-field and stores the card instruction that will be sent to the card in `last_cmd`. The next step the terminal performs is to send a command to the cardlet. The command is made up of the card instruction (`SW_authenticateCard`) and the challenge generated before. After receiving this command the card verifies that all the preconditions of this protocol are fulfilled. First there mustn't be another protocol running, this is checked by testing if `nxt_cmd` is equal to `any` and secondly the user or the conductor must have been authorized. If a precondition isn't satisfied the cardlet generates an appropriat exception. If the test are successful the cardlet stores the received challenge in the field `terminal_challenge` and sends a response to the terminals consisting of `SW_OK` and the signature of the challenge. After receiving the response from the card, the terminal tries to verify the signature. If the signature is correct the terminal sets the field `card_authenticated` to `true` and `last_cmd` to `none`.

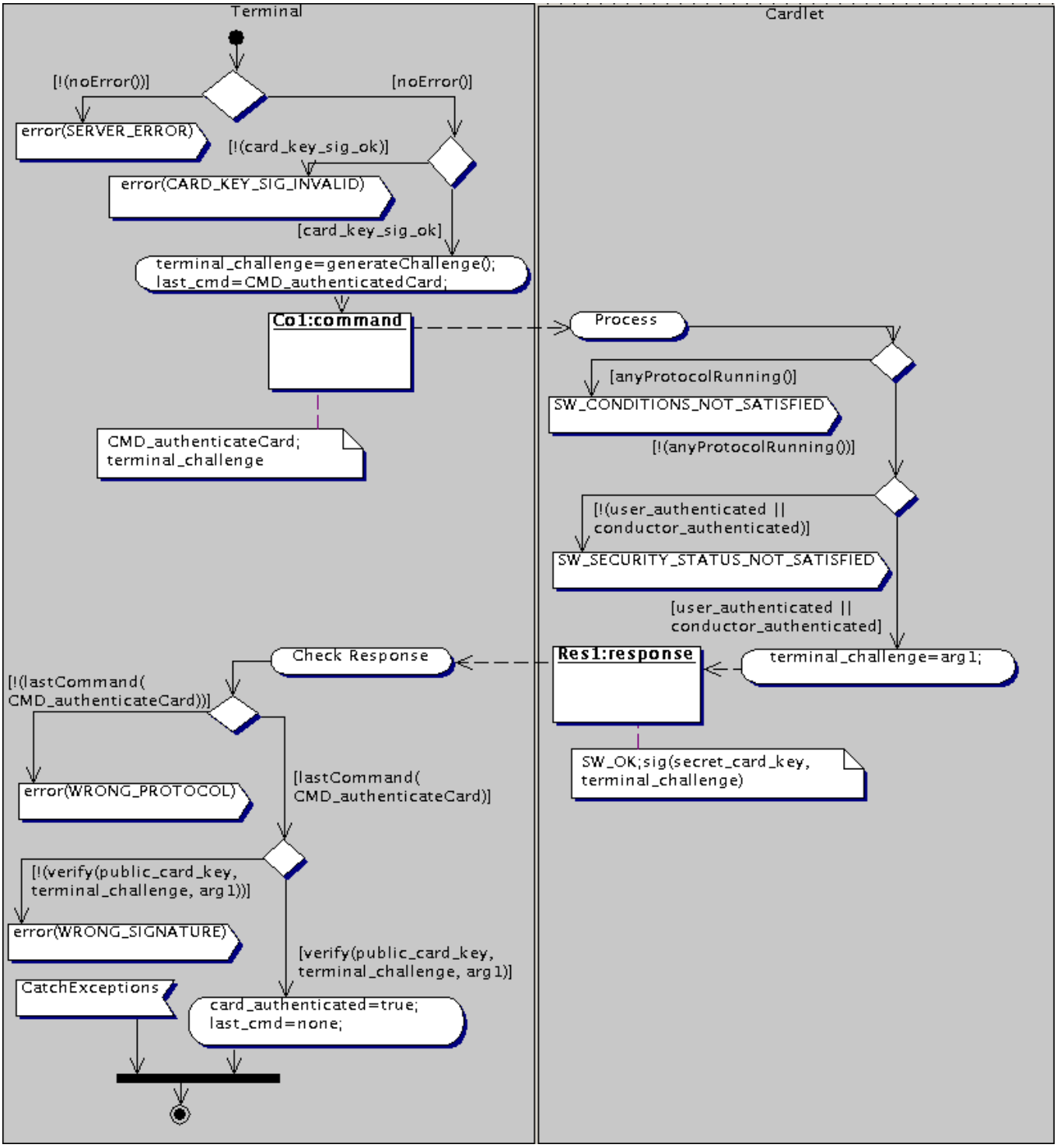


Figure 2.2: *Authenticate Cardlet*-protocol

2.4 Authenticate Conductor

2.4.1 Purpose

This protocol is used to authenticate the conductor terminal. After a successful protocol run the smartcard can be sure to communicate with a genuine conductor terminal.

2.4.2 Preconditions

There mustn't be any other protocol running on the card, i. e. the `nxt_cmd`-field of the cardlet must be `any`.

2.4.3 The protocol

As usual the protocol starts with the terminal checking if it is ready. If there's no problem, the terminal stores the card instruction that will be sent to the cardlet. Then the command, consisting of the card instruction `CMD_loadCondKey` and the public key of the conductor terminal, is sent to the card. The first action to be performed by the card after receiving the command is to check if there's already a protocol running. If this is the case, an exception is generated. Otherwise the cardlet stores the received key in the field `public_cond_key` and the next card instruction (CMD_checkCondKeySig) that must be sent by the terminal in `nxt_cmd`. Thereafter the cardlet send its answer to terminal. The answer consists only of `SW_OK`. After receiving the response form the card, the terminal stores the card instruction that will be sent to the card next (`CMD_checkCondKeySig`) and sends the next command, consisting of the card instruction and the signature of the public key of the conductor (`public_cond_key_sig`), to the card. After receiving this command, the card checks of the card instruction sent by the card is the correct card instruction for the current step in the protocol (`thisProtocolRunning(CMD_checkCondKeySig)`). If the correct card instrucation was sent, the cardlet verifies the signature of the conductor key. If the signature is not correct the cardlet creates an `SW_SECURITY_STATUS_NOT_SATISFIED`-exception, otherwise the cardlet generates a new challenge and stores the next expected card instruction. Then a response is sent to the terminal containing the new challenge. The terminal stores the received challenge and the card instruction that will be sent to the card next and sends a new command to the card. This command consists of the card instruction (`CMD_authenticateConductor`) and the signature of the card challenge. The cardlet first verifies that it is in the correct protocol step and, if this is the case, verifies that the signature of the challenge is correct. If the signature could be verified, the cardlet sets the attribute `conductor_authenticated` to `true` and `next_cmd` to `any`. After receiving the answer, the terminal sets `last_cmd` to `none`.

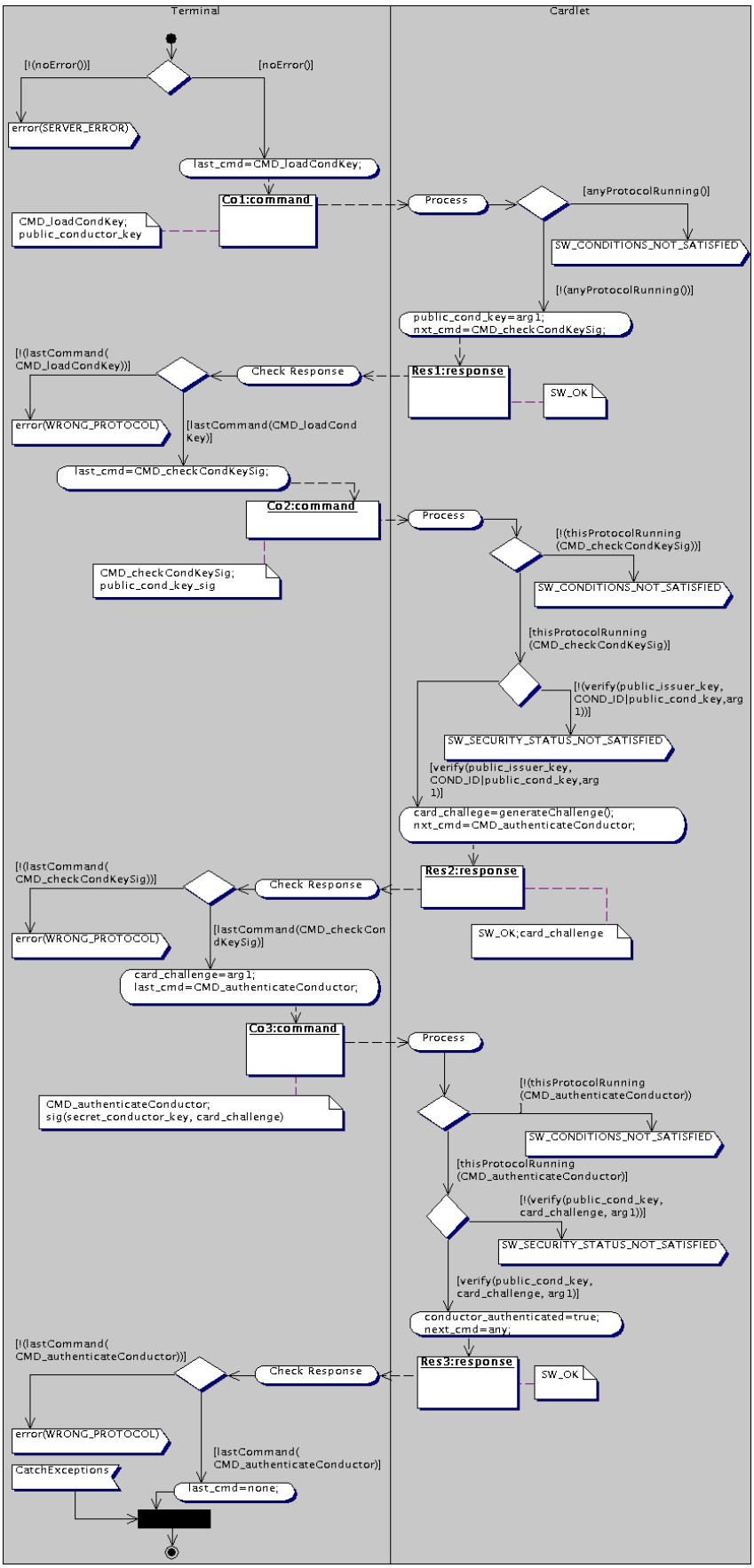


Figure 2.3: *Authenticate Conductor*-protocol

2.5 Authenticate User

2.5.1 Purpose

This protocol is used to verify the authenticity of the cardholder. First the cardholder enters his PIN. Then the inputed value is transmitted to the card which compares it with the expected value.

2.5.2 Preconditions

There mustn't be any other protocol running on the card, i. e. the `nxt_cmd`-field of the cardlet must be `any`.

2.5.3 The protocol

First the terminal checks for internal errors. If the terminal is ready the user is prompted for the PIN. The PIN entered by the user is stored in the attribute pin. The `last_cmd`-attribute is set to `CMD_verifyPIN`. The a command is sent to the cardlet consisting of the card instruction `CMD_verifyPIN` and the PIN entered by the user. After receiving this command the cardlet tests if there isn't a protocol running currently. If there's no other protocol, the cardlet verifies if the transmitted PIN is correct. If this is the case, the attribute `user_authenticated` is set to `true`. `SW_OK` is sent as answer to the terminal. If one of the precondition wasn't satisfied, an appropriate exception is thrown. After receiving the answer, the terminal sets `last_cmd` to `none`.

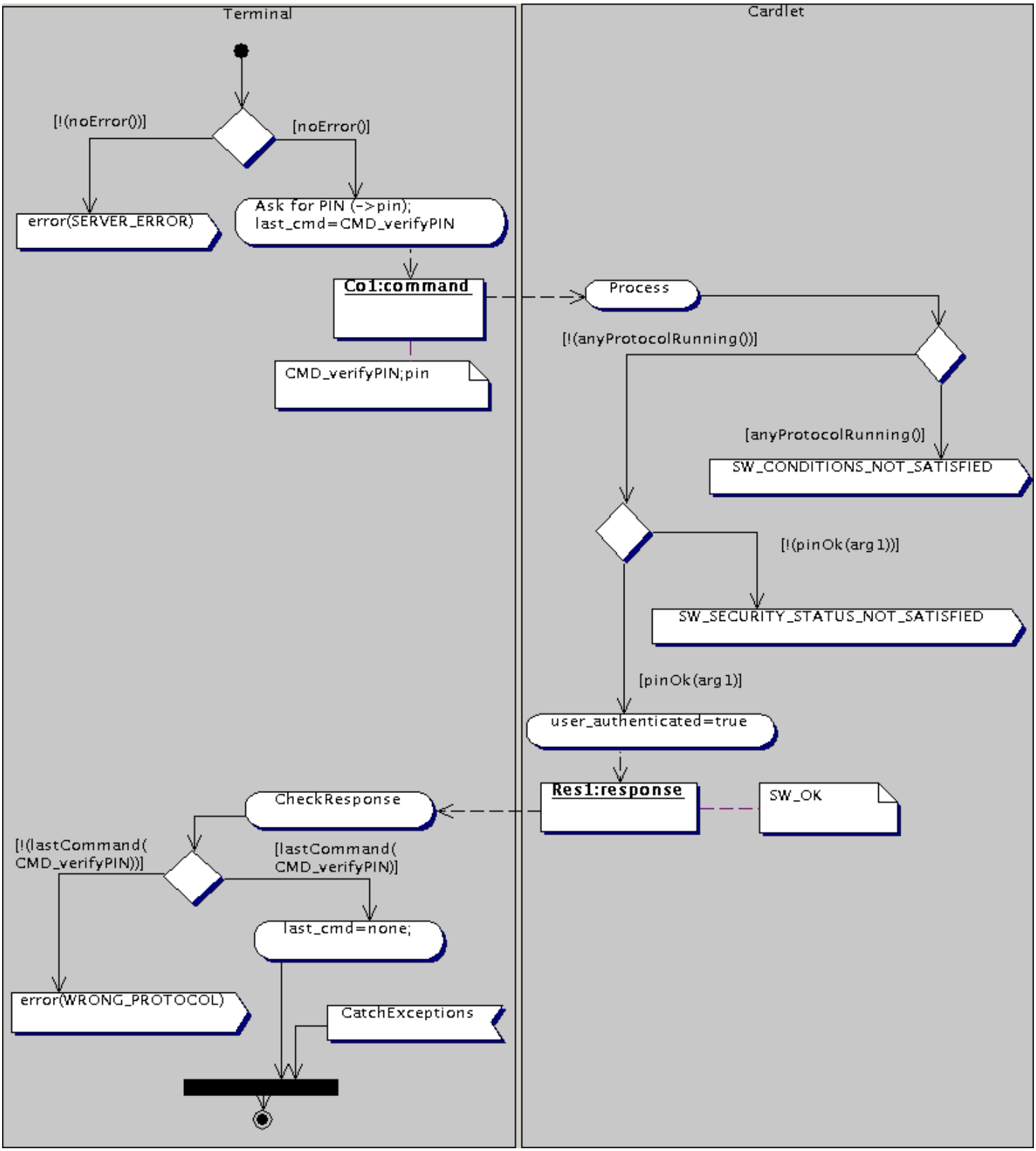


Figure 2.4: *Authenticate User*-protocol

2.6 Load Ticket

2.6.1 Purpose

This protocol is used to load a new ticket on a card.

2.6.2 Preconditions

Before this protocol can be executed the terminal must have acquired and verified the public key of the card. It must be possible to load another ticket on the card, i. e. the number of tickets on the card must be less than `max_tickets`. There mustn't be any other protocol running on the card, i. e. the `nxt_cmd`-field of the cardlet must be `any`.

2.6.3 The protocol

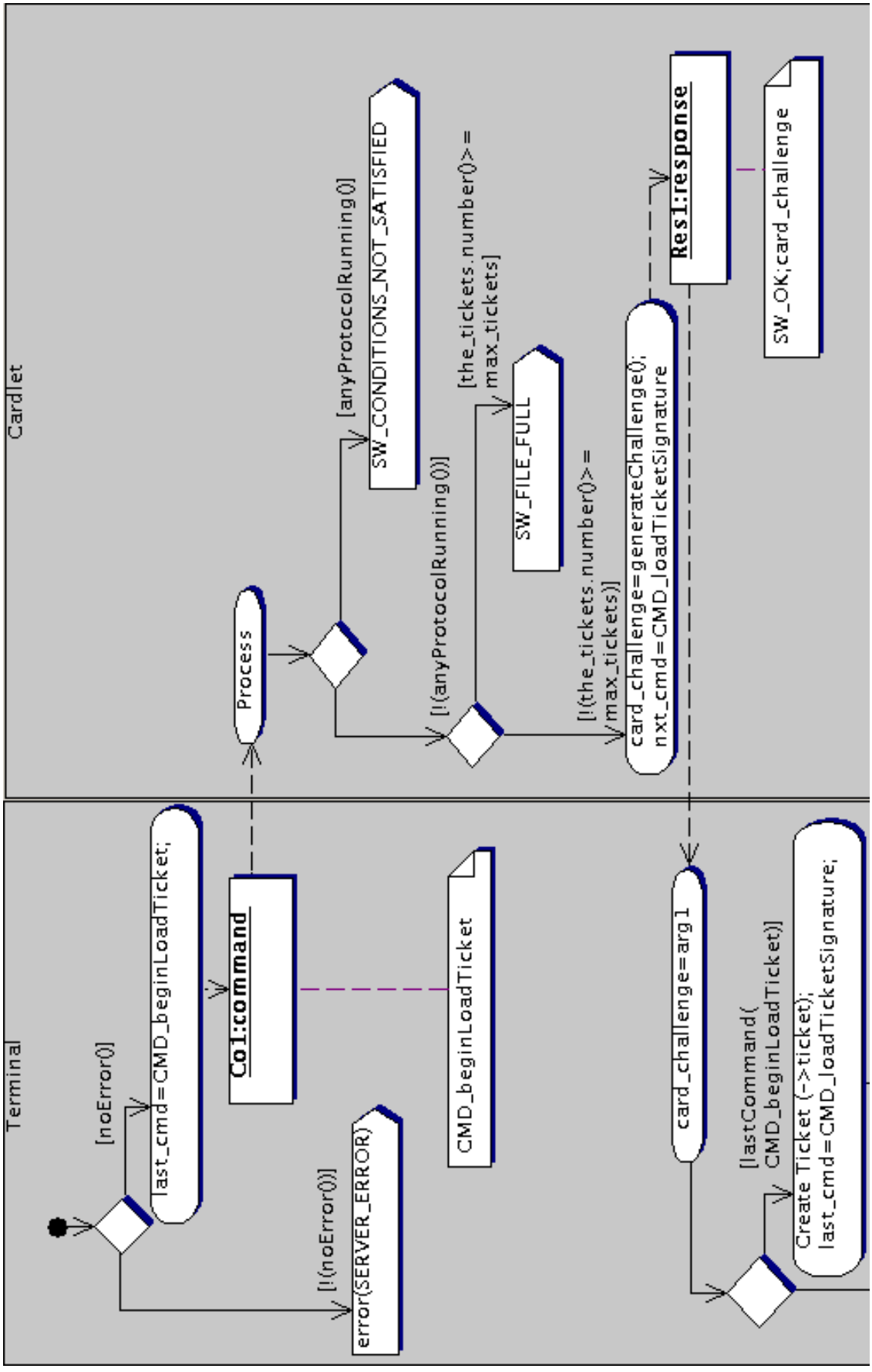


Figure 2.5: *Load Ticket*-protocol Part I

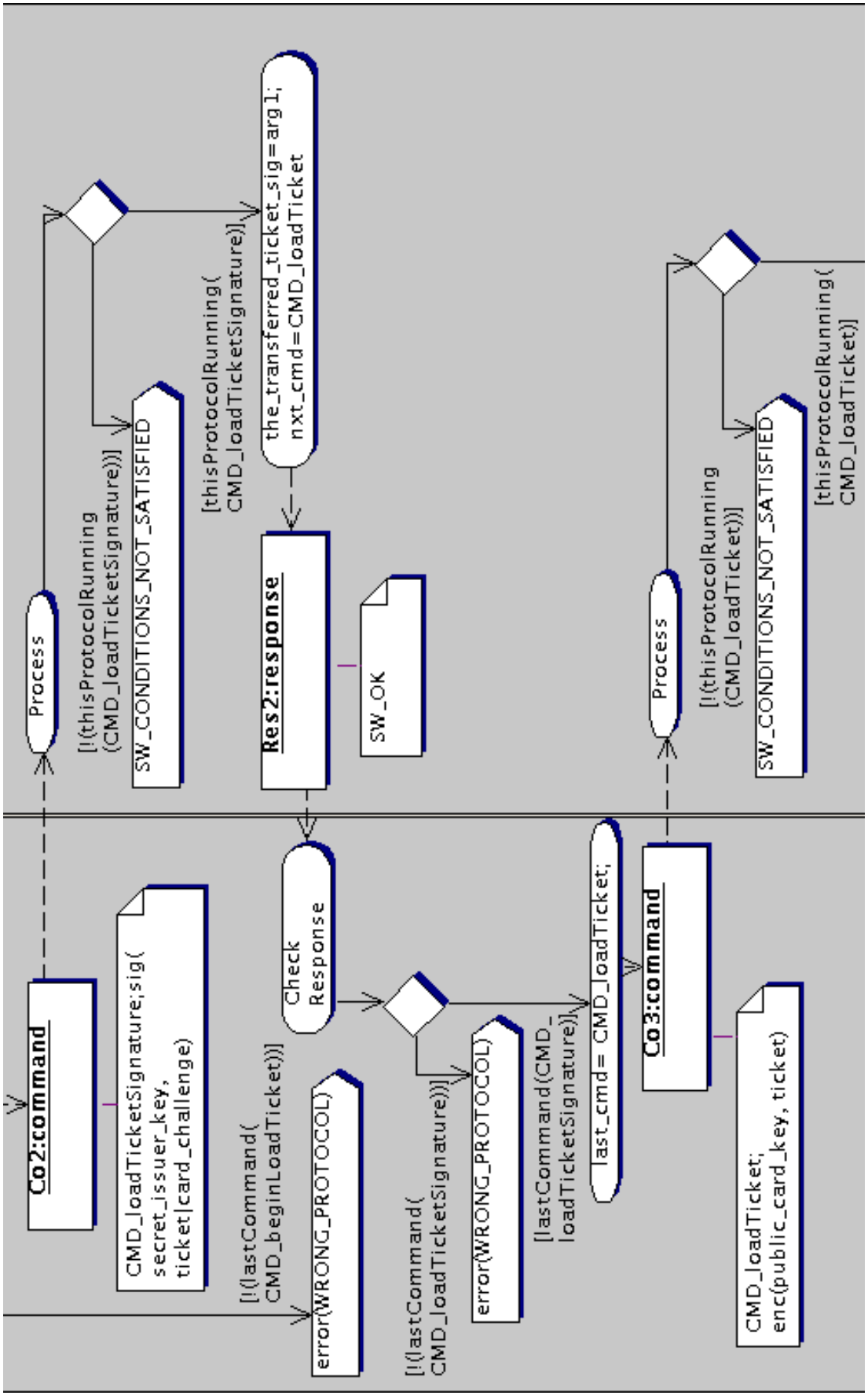


Figure 2.6: *Load Ticket*-protocol Part II

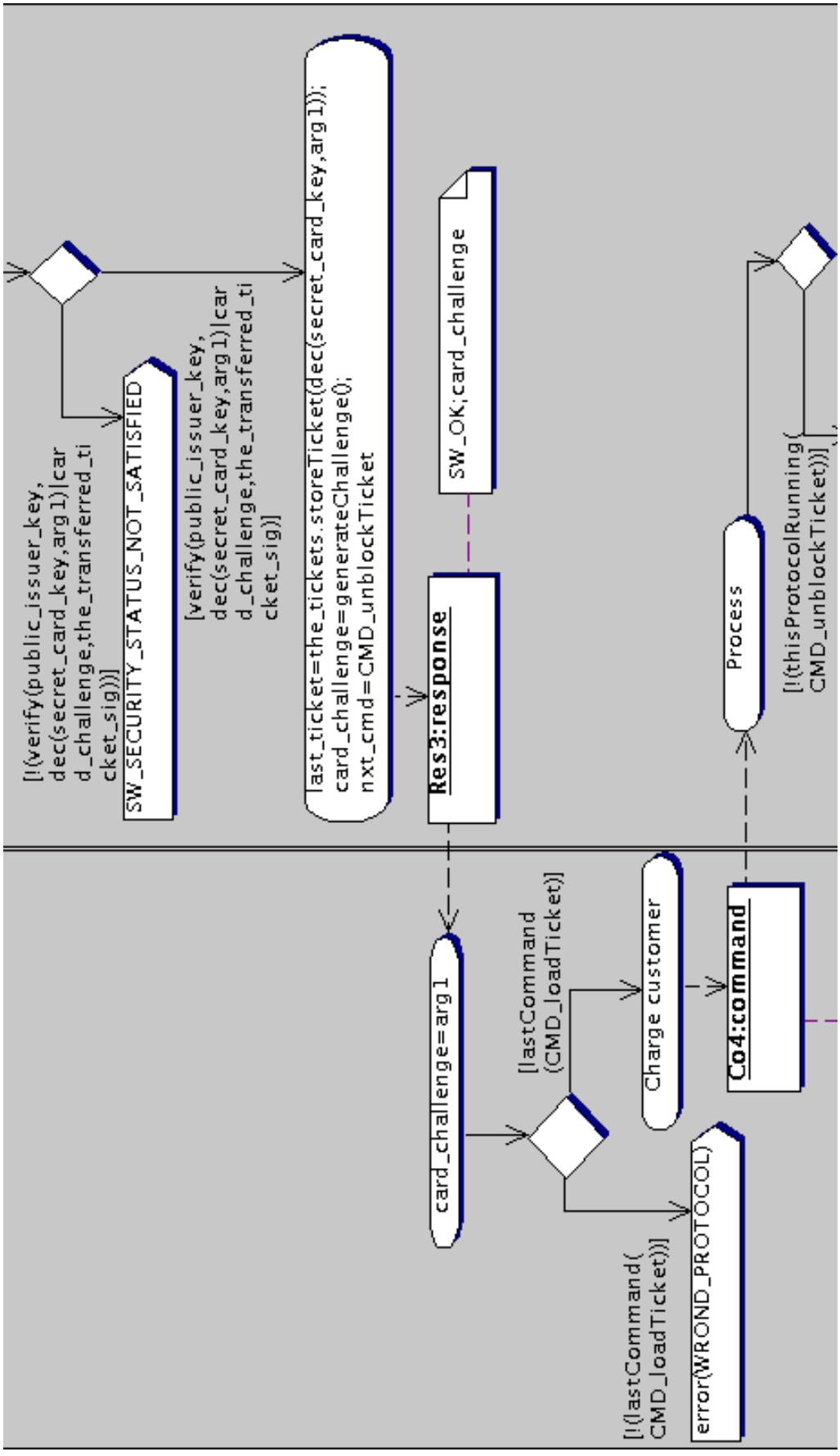


Figure 2.7: *Load Ticket*-protocol Part III

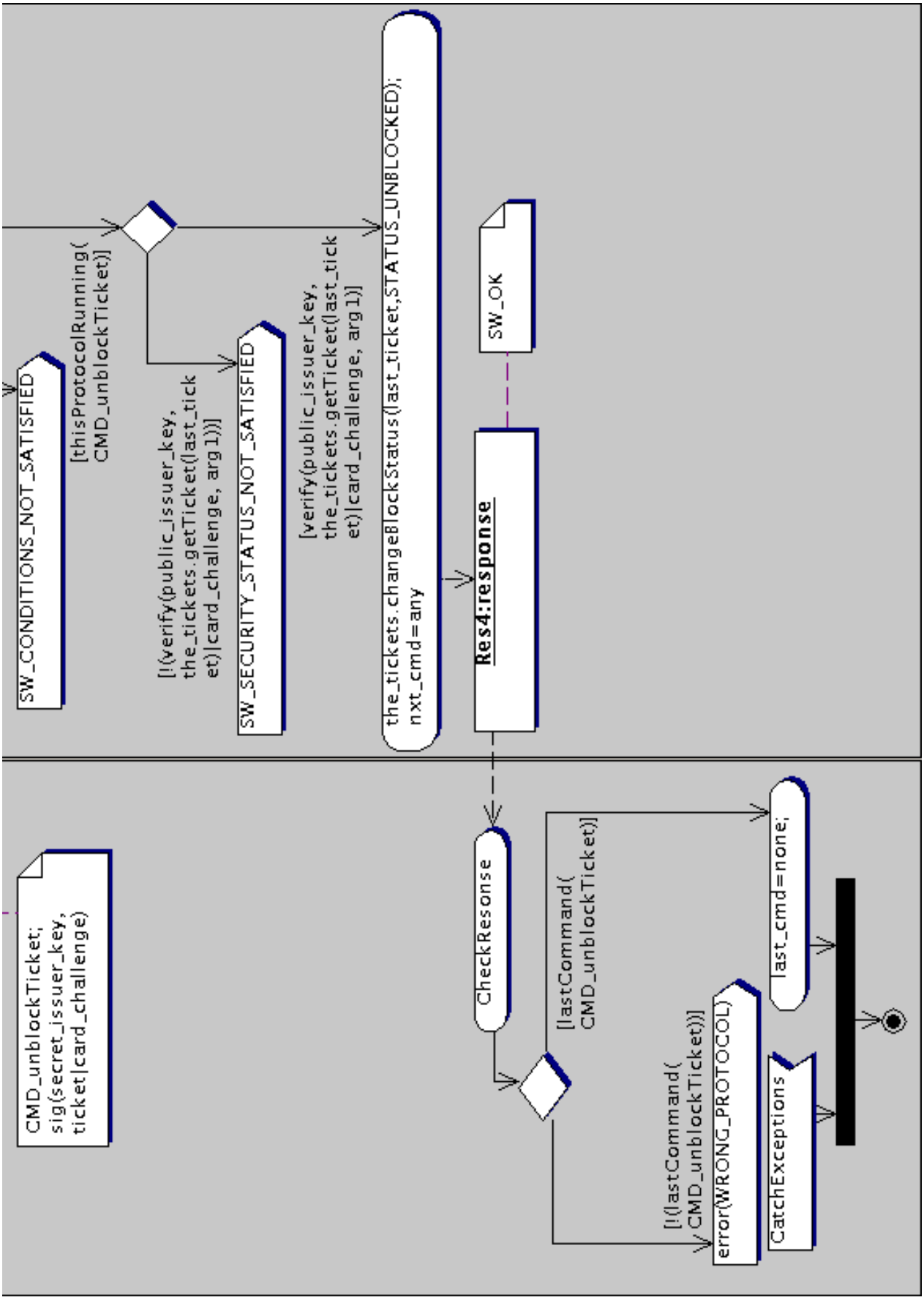


Figure 2.8: *Load Ticket*-protocol Part IV

2.7 Select

2.7.1 Purpose

This protocol describes the actions performed by the `select`-method. The `select`-method of a cardlet is called by the card runtime environment when the card receives an APDU containing a select command for the corresponding cardlet.

2.7.2 Preconditions

None.

2.7.3 The protocol

The `select`-method resets protocol-specific attributes of the cardlet and tries to resume an incomplete ticket transfer if necessary.

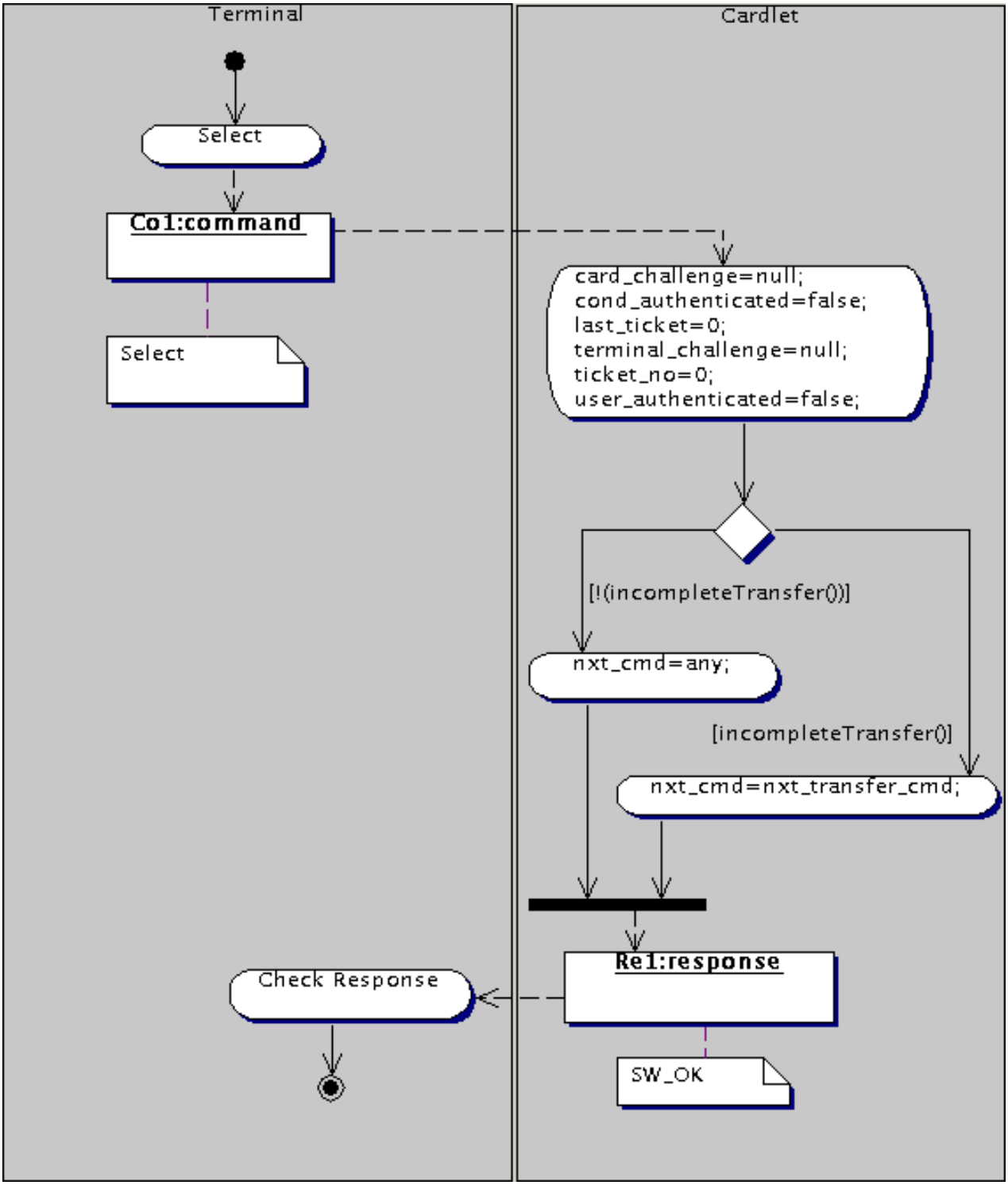


Figure 2.9: *Select*-protocol Part I

Bibliography

[OMG99] The Object Management Group (OMG). *OMG Unified Modeling Language Specification*, 1999. http://www.omg.org/technology/uml.

[RJB98] James Rumbaugh, Ivar Jacobson, and Grady Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1998.