Security System Case Study

Technical Report No.136

Martin Loomes and Carol Britton

March 1992

Security System Case Study

Martin Loomes and Carol Britton
School of Information Science, Hatfield Polytechnic,
College Lane, Hatfield, Herts, AL10 9AB.

March 19, 1992

Abstract

This paper is an early draft of the specification of a security system.

We begin by giving a full description in English of the client company and their requirements for a card issuing system and a door control system. The formal notation, Z, is then used to specify the door control system, in terms of both its basic functionality and its interaction with the people who use it.

The specification of the card issuing system is left as an exercise for the reader. Suggestions as to how to proceed with this can be found at the end of the paper.

System Overview

BLOËM OOP is a scientific research institution which originated in Holland. Since they wish to capitalise on the common European market, the BLOËM OOP management have set up a sister company in Great Britain which is engaged in developing a number of highly sensitive, top secret products. The research and development of each of these products is administered under a separate project and the company usually run several of these projects at any one time. In order to preserve necessary security, the projects are all coded with names such as Athene, Aphrodite and Artemis.

Although BLOËM OOP were not very enthusiastic about the ending of the Cold War, business has in fact been booming. This upsurge of activity has meant that the company has had to increase the number of its staff at all levels of its operation - managers, researchers, scientists, secretaries and even cleaners. The increase in personel, together with the stringent security requirements of their clients, have led BLOËM OOP to consider the installation of a new card-operated door control system to control access to each project room.

Up until now the BLOËM OOP Personnel Department has had a rather casual approach to staff security. Their system keeps track of who is on the staff simply by issuing a pin-on card to each employee. At the moment these cards are the only method of staff identification. It is envisaged, however, that a new, computerised system would use cards with a magnetic strip to control access to secure areas of the building. This new system would be controlled by the Security Department and would only allow a member of staff access to the rooms he has received permission to enter. BLOËM OOP wish to retain the old card issuing procedures as a separate system, but would like the developer to produce an automated version of it.

As an example, the diagram in Figure 1 on the following page shows a ground plan of one floor of the BLOËM OOP building. Staff member Kim Philby has been recruited to work on project Aphrodite. He therefore has permission to enter all the Aphrodite project rooms, the laboratory and the meetings room, but he does not have access to the manager's office. Aphrodite project manager, John Profumo, has access to all the rooms on this floor, but his secretary,Christine Keeler, is only allowed to enter her own office, Mr. Profumo's office and the meetings room.

We are going to describe the required door control system using English and the Z notation. We shall see how this combination of languages allows us to describe precisely what we want the system to do, while at the same time giving us the freedom to explore fully the problem area. The specification of the existing card issuing system is presented as a set of exercises at the end of the paper. These two subsystems can subsequently be combined to form a complete system which can cope with security queries such as "Where is Mr. Profumo?", or "Who is in Aphrodite Room 2?".

Interviews have taken place with the head of the Personnel Department to ascertain how the existing manual card issuing system is organised, and with the head of the Security Department to establish requirements for the new door control system. Extracts from the developer's notes from both these interviews can be found in the next section.

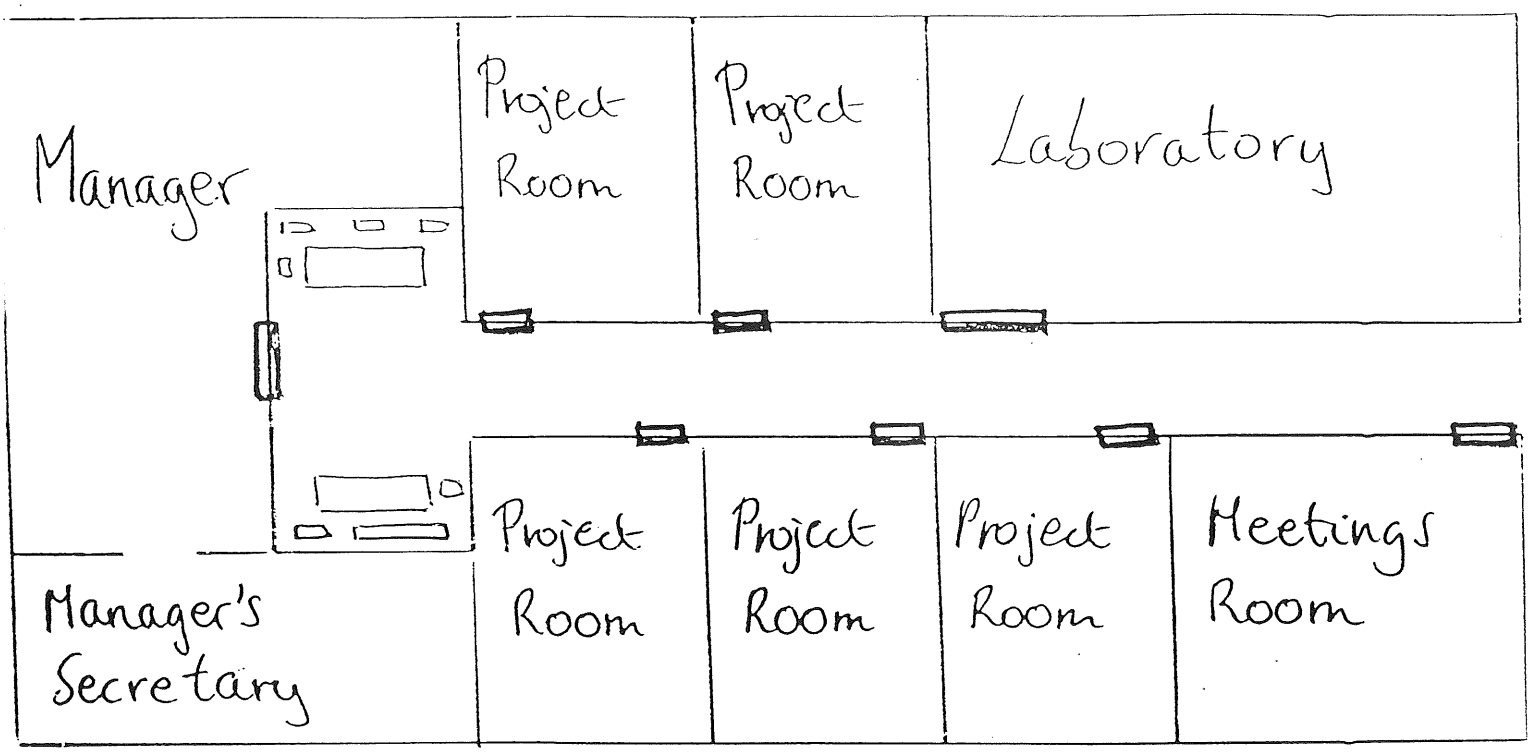

Figure 1. Ground plan of one floor of the BLOËM OOP building.

Interview Notes

Interview with Guy Burgess, Head of Personnel, BLOËM OOP.

- Cards are bought in from outside suppliers in batches of 100.
- A member of the Personnel Department checks that all the new cards have a unique number and that none of these numbers coincide with any already in use at BLOËM OOP. This is done by checking the new card numbers against a master list of staff members, their departments and their card numbers. Names are assumed to be unique within departments and this is enforced, if necessary, by the adoption of fictitious middle initials.
- A list is also held of all card numbers which are still available for allocation.
- As each new card is given to a staff member, its number is recorded on the master list together with the name and department of the staff member to whom it has been allocated. No member of staff ever has more than one card.
- Any member of staff who loses a card must inform the Personnel Department who will check the card number on the master list and mark it as 'Lost'. A replacement card can then be issued.
- A member of staff who leaves the company is supposed to hand in the card to the Personnel Department, so that it can be re-issued to another employee.In practice most members of staff forget to do this and so their cards are simply recorded as lost.
- The card issuing system is run entirely by the Personnel Department who perform all administrative functions connected with the system.

Interview with Donald McClean, Head of Security, BLOËM OOP.

- The Security Department will be in control of the new security system. Since the Personnel Department already deals with the issuing of cards to members of staff, the new system will simply keep track of cards, not people.
- The Security department will add permissions to enter certain rooms to each card. The system will only need to know which permissions are allowed on any card, but the Security Department will have to know which member of staff the card relates to, so that the correct permissions can be allocated. Each time a card is allocated to a member of staff the Security Department will have to be informed of the card number and the name and department of the employee to whom it has been allocated.
- Cleaners will have to be treated differently from other employees. Currently cleaners only knows which rooms they'll be working in on a particular day when they report for duty in the morning. This should be maintained for security reasons, so cleaners will not have any permanent permissions to enter rooms in the new system. The system will have to have some way of dealing with the problem of temporary permissions.
- Each card should be able to serve as a pass to open doors to secure areas, provided that the member of staff with the card has permission to enter the area.

- As well as knowing if a new card is allocated to any member of staff, the security system will have to know if a card is cancelled for any reason, or if it is subsequently issued to a different employee.
- The security controller will have to be given appropriate functions to administer the system, such as adding and removing permissions, and entering and deleting card numbers from the system.
- Members of staff can only attempt the two functions of entering and leaving a room. They cannot carry out any administrative functions.
- To enter a room an employee must have a valid card. The system should then check that the employee has permission to enter the room and that he or she is not already recorded as inside a secure area. To do this the system will have to keep track of which employee is in which room at any one time.
- Once a staff member has inserted a card and all details have been checked and found to be in order, the system will release an automatic locking mechanism on the door and register the card as inside the room. This assumes that when a card is used to open a door the cardholder enters the room, taking the card along.
- To leave a room, an employee will have to insert a valid card. The system will then open the door and log the card out of the room. The system must only open the door, however, if the card being used is currently logged as being in the room being left. This means that anyone who gets into a secure area by slipping in with a cardholder will only be able to get out in the same way.
- Obviously all locking mechanisms must be automatically released in the case of a fire alarm. A master switch will be needed to ensure this, and the security controller will have to have an administrative function to reset the system before employees are allowed back in. Of course the computer system will not be able to check that each room is actually empty, so it is assumed that, as far as possible, the security guards will check all rooms to make sure that everyone has left and lock each room after the check.
- Certain functions will be required which can only be achieved by combining the card issuing and the security systems. These include finding the whereabouts of a particular member of staff (not just the card) and finding out which employees are in a particular room.

1 The Door Control System

To produce a clear, well-structured specification we need to separate the essential functionality of the system (the checking of cards in and out of secure rooms) from its interface (the way in which it reacts with its users). This separation of concerns brings us two advantages: first,we will have a better understanding of the different components of the system; second, we have the ability to modify the interface at a later date without affecting the basic functionality of the system.

We start by defining the data types that we are going to use to capture the values used in the system, then we define the user interface and the effect of the externally visible operations in

terms of changes to stored variables of the defined types. Finally we define a number of functions for manipulating these values.

1.1 The Data Model

To specify the Door Control system we need to use the types *Card* and *Room* We leave both the types free, as we are not bothered about the exact format of these values at present.

At any given moment the system will have to know certain facts;

- which cards are registered as part of the door control system
- which cards (and therefore employees) are allowed in which rooms
- which cards (employees) are at present in a secure room

To keep track of which cards are in which rooms we will use a partial function: partial because it is possible that not all cards will be in rooms at any given time. We use a function, rather than a relation because we assume that each employee can only be in one room at a time. This is important, because it can be used to trap security breaches where cards are slipped under doors to allow illicit entry to rooms.

$$In \mathrel{\widehat=} Card \pfun Room$$

This function can tell us, for example, that Card 4502 is in Room 3. However we still have to revert to the manual card issuing system to discover which employee this card belongs to.

To record who is allowed into which project rooms we will use another partial function, one that maps a card to a set of rooms that the cardholder is entitled to enter. We have decided that all cards known (that is, all those registered by the security controller) will have an entry in the function, but some may have an entry indicating that the cardholders are not allowed into any rooms. This may seem strange, but it simplifies our specification, for this function can now also be used to keep track of which cards are known to the system.

It is important to note that this is a decision we have made in writing the specification, but the implementor of the system may well decide to program this without keeping null records for employees with no permissions. This is a good example of one of the claimed advantages of using a formal notation as a specification tool - that it leaves the programmer with the maximum amount of freedom for the implementation.

We define the function which records which cards are allowed into which rooms as follows :-

$$Permissions \mathrel{\widehat=} Card \pfun \mathbb{P}\, Room$$

We will eventually require a variable to hold all this information, so we define a type corresponding to values which capture both the record of which card is in which room, and also what permissions all cardholders employees have. This is simply a tuple of the two functions described above.

$$State \mathrel{\widehat=} In \times Permissions$$

Some of our system operations will only change one part of this state, so it is convenient to have functions that allow us to extract just the first or second component of the state tuple. We will call these two functions *inComponent* and *permissionComponent* respectively. These are defined as follows:

STATE_TYPE_SELECTORS

$$inComponent : State \rightarrow In$$
$$permissionComponent : State \rightarrow Permissions$$

$$inComponent(i, p) = i$$
$$permissionComponent(i, p) = p$$

1.2 The State Variable

We are now going to pave the way for describing our externally visible operations. To do this we need to describe how the state actually changes as we perform the operations, and any responses the system makes. This is fundamentally different from a purely functional approach, such as is used in OBJ, where a function returns a new value each time it is invoked. In defining a state variable, we are recognising the existence of information which is stored inside the system and which will be modified by the operations which are performed on the system.

To define the state variable we will introduce two values of type state, so that we can describe the operations in terms of changes of state.

ΔSTATE_VARIABLE

$$STATE\_TYPE\_SELECTORS$$
$$st, st' : State$$

We can extend this schema to introduce an invariant on the state (something that our operations must not be allowed to violate). We will insist that cardholders are not allowed to be in rooms that they do not have permissions to be in.

ΔSTATE_VARIABLE

$$STATE\_TYPE\_SELECTORS$$
$$st, st' : State$$

$$\forall c : Card \bullet (inComponent\ st\ c) \in (permissionComponent\ st\ c)$$
$$\qquad \wedge\ (inComponent\ st'\ c) \in (permissionComponent\ st'\ c)$$

1.3 The Outer Layer - The System Interface

We can now define our user interface. This involves specifying the behaviour of the system, in terms of outputs and changes of state, corresponding to the invocation of operations with

particular inputs.

The operations we need to provide are those to add and delete cards from the system, reset after an alarm, allow (or deny) entry and exit. For each of the possible operations we will provide a case analysis that describes the behaviour of the operation under all possible situations. We will also provide a response to the user, so that acknowledgements for actions, or messages saying why the operations won't behave the way the user expected, can be provided.

$$Response \mathrel{\widehat=} \{\text{"}Card\ added\text{"}, \text{"}Card\ not\ known\text{"}, \text{"}Permission\ denied\text{"}, \text{"}Duplicate\ card\text{"}, \\ \text{"}Card\ in\ secure\ room\text{"}, \text{"}Card\ deleted\text{"}, \text{"}System\ reset\text{"}, \\ \text{"}Door\ unlocked\text{"}, \text{"}Card\ is\ already\ in\ secure\ room\text{"}, \\ \text{"}Card\ is\ not\ in\ secure\ room\text{"}\}$$

First we will define the administration operations provided for the security controller.

1.3.1 The Add Card Operation

This operation defines the system's reaction to an attempt to add a new card.

CASE 1 Card already known by the system.
Output the response "Duplicate Card", and do nothing

CASE 2 Card currently not known to the system.
Output the response "Card Added" and add the card to the system.

This is formalised by

INPUTS $c?$, the card being added

OUTPUTS $resp!$, the system response.

CASE 1 $c? \in dom(permissionComponent\ st)$
$resp! = \text{"}Duplicate\ card\text{"} \land st' = st$

CASE 2 $c? \notin dom(permissionComponent\ st)$
$resp! = \text{"}Card\ added\text{"} \land st' = addCard(st, c?)$

This is reflected by the schema

$$
\begin{array}{|l}
\textit{ADD\_CARD\_OPERATION} \\
\Delta \textit{STATE\_VARIABLE} \\
\textit{ADD\_CARD\_FUNCTION} \\
c? : \textit{Card} \\
\textit{resp}! : \textit{Response} \\
\hline
\quad (c? \in \textit{dom}(\textit{permissionComponent}\ st) \\
\quad \land \\
\quad \textit{resp}! = \text{“}\textit{Duplicate card}\text{”} \\
\quad \land \\
\quad st' = st) \\
\lor \\
\quad (c? \notin \textit{dom}(\textit{permissionComponent}\ st) \\
\quad \land \\
\quad \textit{resp}! = \text{“}\textit{Card added}\text{”} \\
\quad \land \\
\quad st' = \textit{addCard}(st, c?)) \\
\hline
\end{array}
$$

At this stage we have not yet defined the function *addCard*. This is because we are concentrating here on the outer layer of the system - its interface with its users and the way it reacts under various conditions. For the moment all we need be concerned about is that a successful attempt to add a new card to the system brings about a new system state. This new state is the result of applying a function which we have called *addCard* to the old system state and the new card which we wish to add to it. We can leave consideration of how *addCard* actually works until later in the specification.

1.3.2 The Delete Card Operation

This operation deals with the case where we want to remove a card from the system, such as when an employee looses a card, or leaves the company without returning the card. Note that if we delete a card when the cardholder is currently in a room we have a card in a room which is not officially known to the system. We will guard against this by ensuring that we only delete cards for employees who are currently not in any room.

CASE 1 The card is not known by the system.
Output the response "Card not known", and do nothing

CASE 2 The card is known.

CASE 2.1 The card is registered as currently in a room.
Output "Card in secure room", and do nothing. (We do not want to delete a card from the system while it is inside the secure area).

CASE 2.2 The card is not registered as currently in a room.
Output "Card deleted" and delete the card

Formalised, this becomes

CASE 1 $c? \notin \mathrm{dom}(permissionComponent\ st\ )$
$resp! = \text{“}Card\ not\ known\text{”} \land st' = st$

CASE 2 $c? \in \mathrm{dom}(permissionComponent\ st)$

CASE 2.1 $c? \in \mathrm{dom}(inComponent\ st)$
$resp! = \text{“}Card\ in\ secure\ room\text{”} \land st' = st$

CASE 2.2 $c? \notin \mathrm{dom}(inComponent\ st)$
$resp! = \text{“}Card\ deleted\text{”} \land st' = deleteCard(st, c?)$

This is reflected by the schema

DELETE_CARD_OPERATION

$\Delta STATE\_VARIABLE$
$DELETE\_CARD\_FUNCTION$
$c? : Card$
$resp! : Response$

$(c? \notin \mathrm{dom}(permissionComponent\ st)$
$\land$
$resp! = \text{“}Card\ not\ known\text{”}$
$\land$
$st' = st)$
$\lor$
$(c? \in \mathrm{dom}(permissionComponent\ st)$
$\land$
$((c? \in \mathrm{dom}(inComponent\ st)$
$\land$
$resp! = \text{“}Card\ in\ secure\ room\text{”}$
$\land$
$st' = st)$
$\lor$
$(c? \notin \mathrm{dom}(inComponent\ st)$
$\land$
$resp! = \text{“}Card\ deleted\text{”}$
$\land$
$st' = deleteCard(st, c?))))$

1.3.3 The Operation to Reset the System After an Alarm

There is no need for a case analysis here, for the operation must behave the same in all circumstances. The system must clear the record of which cardholders are in which rooms, but leave the permissions record unchanged.

EVACUATE_BUILDING_OPERATION

$\Delta STATE\_VARIABLE$
RESET_AFTER_ALARM_FUNCTION
$resp! : Response$

$resp! = \text{"}System\ reset\text{"}$
$\wedge$
$st' = reset(st)$

1.3.4 Entering a Room

When an employee attempts to enter a room, several things may go wrong. First, the card being used may not be recognised by the system. Second, the card may not have permission for the room being entered. Finally, the employee may already be in a room (as far as the system knows). This situation might arise if an employee passes a card out through a window to an accomplice, or if an employee sneaks out through a door without using a card (eg. with someone else).

We assume that the output of a response "Door unlocked" here is accompanied by the physical unlocking of the door, and other messages may be accompanied by alarm bells, messages to the security controller, or whatever is deemed appropriate.

CASE 1 Card not known to the system
Output the mesage "Card not known" and do nothing.

CASE 2 Card is known to the system

CASE 2.1 The card does not have permission for the room being entered.
Output "Permission denied", and do nothing

CASE 2.2 Permission is held for the room being entered.

CASE 2.2.1 The card is already logged as being in a room
Output "Card is already in secure room" and do nothing

CASE 2.2.2 The card is not logged as already in a room
Output "Door unlocked" and admit the employee.

INPUTS $c?$, the card being used
$r?$, the room being entered.

OUTPUTS $resp!$, the system response

CASE 1 $c? \notin \text{dom}(permissionComponent\ st)$
$resp! = \text{"}Card\ not\ known\text{"} \wedge st' = st$

CASE 2 $c? \in \text{dom}(permissionComponent\ st)$

CASE 2.1 $r? \notin (permissionComponent\ st\ c?)$
$resp! = \text{"}Permission\ denied\text{"} \wedge st' = st$

CASE 2.2 $r? \in (permissionComponent\ st\ c?)$

CASE 2.2.1 $c? \in \mathrm{dom}(inComponent\ st)$
$resp! = \text{“}Card\ is\ already\ in\ secure\ room\text{”} \land st' = st$

CASE 2.2.2 $c? \notin \mathrm{dom}(inComponent\ st)$
$resp! = \text{“}Door\ unlocked\text{”} \land st' = enterRoom(st, c?, r?)$

This becomes the schema

ENTRY_REQUEST

$\Delta STATE\_VARIABLE$
$ENTER\_ROOM\_FUNCTION$
$r? : Room$
$c? : Card$
$resp! : Response$

$(c? \notin \mathrm{dom}(permissionComponent\ st)$
$\land$
$resp! = \text{“}Card\ not\ known\text{”}$
$\land$
$st' = st)$
$\lor$
$(c? \in \mathrm{dom}(permissionComponent\ st)$
$\land$
$(r? \notin (permissionComponent\ st\ c?)$
$\land$
$resp! = \text{“}Permission\ denied\text{”}$
$\land$
$st' = st)$
$\lor$
$(r? \in (permissionComponent\ st\ c?)$
$\land$
$(c? \in \mathrm{dom}(inComponent\ st)$
$\land$
$resp! = \text{“}Card\ is\ already\ in\ secure\ room\text{”}$
$\land$
$st' = st)$
$\lor$
$(c? \notin \mathrm{dom}(inComponent\ st)$
$\land$
$resp! = \text{“}Door\ unlocked\text{”}$
$\land$
$st' = enterRoom(st, c?, r?))$

1.3.5 Leaving a Room

This operation is similar to that for entering a room, but this time we will not check permissions, but only check that the cardholder is in the room being left. Strictly we do not need to check that the card is recognised, for the cardholder cannot have entered the room if the card is not recognised, and the card cannot be removed from the system while a cardholder is in a secure room. We have chosen to leave it in, however, to handle cases where cards become invalid whilst a cardholder is in a room.

As with the entry requests, we assume that messages may be accompanied by suitable actions such as unlocking doors.

CASE 1 Card not known to the system
Output the mesage "Card not known" and do nothing.

CASE 2 Card is known to the system

CASE 2.1 Card is not logged into room being left.
Output "Card is not in secure room", and do nothing

CASE 2.2 Card is logged into room being left.
Output "Door unlocked" and log the cardholder as having left the room.

Formally,

CASE 1 $c? \notin \mathrm{dom}(permissionComponent\ st)$
$resp! = \text{"}Card\ not\ known\text{"} \wedge st' = st$

CASE 2 $c? \in \mathrm{dom}(permissionComponent\ st)$

CASE 2.1 $r? \neq (inComponent\ st\ c?)$
$resp! = \text{"}Card\ is\ not\ in\ secure\ room\text{"} \wedge st' = st$

CASE 2.2 $r? = (inComponent\ st\ c?)$
$resp! = \text{"}Door\ unlocked\text{"} \wedge st' = leaveRoom(st, c?)$

$$
\begin{array}{|l}
\hline
\textit{EXIT\_REQUEST} \\
\Delta \textit{STATE\_VARIABLE} \\
\textit{LEAVE\_ROOM\_FUNCTION} \\
r? : \textit{Room} \\
c? : \textit{Card} \\
\textit{resp}! : \textit{Response} \\
\hline
\quad (c? \notin \mathrm{dom}(\textit{permissionComponent}\ st) \\
\quad \land \\
\quad \textit{resp}! = \text{“}\textit{Card not known}\text{”} \\
\quad \land \\
\quad st' = st) \\
\lor \\
\quad (c? \in \mathrm{dom}(\textit{permissionComponent}\ st) \\
\quad \land \\
\qquad\qquad (r? \neq (\textit{inComponent}\ st\ c?) \\
\qquad\qquad \land \\
\qquad\qquad \textit{resp}! = \text{“}\textit{Card is not in secure room}\text{”} \\
\qquad\qquad \land \\
\qquad\qquad st' = st) \\
\qquad \lor \\
\qquad\qquad (r? = (\textit{inComponent}\ st\ c?) \\
\qquad\qquad \land \\
\qquad\qquad \textit{resp}! = \text{“}\textit{Door unlocked}\text{”} \\
\qquad\qquad \land \\
\qquad\qquad st' = \textit{leaveRoom}(st, c?))) \\
\hline
\end{array}
$$

1.4 The Inner Layer - Functions to Manipulate the State Variable

Now that we have specified the ways in which the system will react to various situations, we need to define a number of functions to manipulate values of type *State*. These functions should not be confused with the *operations* that we have already defined to provide our user interface. Our definition of this interface prescribes under what conditions the functions can be invoked. Because we have already specified this, we do not need to worry about imposing preconditions on our functions at this stage. All of the functions make sense without preconditions, it is only that their *use* may be inappropriate in certain situations. This technique of separating the functions from the different ways in which they may be used will allow maximal re-use of the functions at a later stage.

First we define the function *addcard* which is called in the Add Card Operation specified above. This function adds an employee's card to the state variable. We have decided that a new card starts with no permissions. These can be added later if necesary.

ADD_CARD_FUNCTION

$$
\begin{array}{l}
STATE\_TYPE\_SELECTORS \\
addCard : State \times Card \rightarrow State \\
\hline
addCard(st, c) = (inComponent\ st, permissionComponent\ st \oplus \{c \mapsto \{\}\})
\end{array}
$$

We also want to define a function to delete a card from the system. In this case we delete the card's entry from the permissions component of the state, thus we model the validity of a card purely by whether or not it has an entry in the permission component of the state.

DELETE_CARD_FUNCTION

$$
\begin{array}{l}
STATE\_TYPE\_SELECTORS \\
deleteCard : State \times Card \rightarrow State \\
\hline
deleteCard(st, c) = (inComponent\ st, c \lhd permissionComponent\ st)
\end{array}
$$

Next we want to add permissions to a card. We have decided to add just one permission at a time.

ADD_PERMISSION_FUNCTION

$$
\begin{array}{l}
STATE\_TYPE\_SELECTORS \\
addPermission : State \times Card \times Room \rightarrow State \\
\hline
addPermission(st, c, r) = \\
\quad (inComponent\ st, permissionComponent\ st \oplus \\
\qquad \{c \mapsto (permissionComponent\ st\ c \cup \{r\})\})
\end{array}
$$

We also need to be able to remove permissions from a card

DELETE_PERMISSION_FUNCTION

$$
\begin{array}{l}
STATE\_TYPE\_SELECTORS \\
deletePermission : State \times Card \times Room \rightarrow State \\
\hline
deletePermission(st, c, r) = \\
\quad (inComponent\ st, permissionComponent\ st \oplus \\
\qquad \{c \mapsto (permissionComponent\ st\ c \setminus \{r\})\})
\end{array}
$$

Next we will provide a function to reset the system after a fire alarm. This simply preserves all the permissions, but logs everyone out of all rooms.

RESET_AFTER_ALARM_FUNCTION

$$
\begin{array}{l}
STATE\_TYPE\_SELECTORS \\
reset : State \rightarrow State \\
\hline
reset(st) = (\{\}, permissionComponent\ st)
\end{array}
$$

Finally we have two functions to calculate new values of type *State* that result from entry to a room, or exit from a room.

If a cardholder gains access to a room we must add a new entry to the In component of the state. Note that we have used function overriding, not union. Strictly we only need union, as the cardholder cannot already be in the room in question. But again, this is only known as a result of the context in which the function is to be used, so we have kept the function as general as possible by using override. As we shall see, this permits extensions to the system to be accomplished with a minimum of extra work.

ENTER_ROOM_FUNCTION

$$STATE\_TYPE\_SELECTORS$$

$$enterRoom : State \times Card \times Room \rightarrow State$$

$$enterRoom(st, c, r) = (inComponent\ st \oplus \{c \mapsto r\}, permissionComponent\ st)$$

When a cardholder leaves a room, we simply remove the card from the domain of the In component. Since the employee can only be in one room at a time, we don't need to specify which room is being left.

LEAVE_ROOM_FUNCTION

$$STATE\_TYPE\_SELECTORS$$

$$leaveRoom : State \times Card \rightarrow State$$

$$leaveRoom(st, c) = (c \lhd inComponent\ st, permissionComponent\ st)$$

1.5 What have we achieved so far?

This completes the draft specification of the BLOËM OOP door control system. Almost all the points raised in the interview with the head of security of the company have been covered, although we have not explicitly addressed the question of access for cleaning staff. We will assume that there is a master card which the security controller can use to allow cleaners to enter secure rooms as necessary. Functions such as identifying who is in a particular room, which require the combination of the door control and card issuing systems, cannot, of course, be defined formally until the card issuing system has been specified. Guidelines on how to proceed with this can be found in section 1.6.

The door control specification was developed in two distinct parts - the outer layer or interface of the system, and the inner layer or system functionality. This separation has several advanatges. It breaks down the problem area into smaller, more manageable sections, thus giving us a more thorough understanding of the system during the development of the specification. Any problem has to be decomposed into 'brain-sized chunks' before we can start to tackle it and disconnecting the interface from the functionality of the system is a useful and effective way of achieving this problem decomposition. The separation of the specification into distinct components simplifies modification of it at later stages, since we can alter the way in which we have defined either the

functions or the operations on the system without affecting the other component. As development of a system progresses modification is always necessary. Good decomposition of the problem means that we can avoid massive rewrites to accommodate small changes and reuse the parts of our specification which are still relevant.

1.6 Guidelines for the specification of the Card Issuing System

At any given time the card issuing system needs to know the following facts :-

- which cards are available for allocation
- which card has been allocated to which employee. We assume here that at any time an employee can only have one card and that a card can only be allocated to one employee. BLOËM OOP have indicated that they wish all employees to be identified by their name and department, so the system will have to hold this information as well.

Exercise 1 Model the system components *Available* and *Allocated* as described above.

The state of the card issuing system is now captured as a tuple of these two components.

$$State == Available \times Allocated$$

Exercise 2 Define functions that will allow you to extract and modify just one component of the system tuple. Call these functions *AvailableComponent* and *AllocatedComponent*. You can see how to do this In section 1.1 which describes the data model of the door control system.

Exercise 3 Define a schema to introduce the state variable. You can see how to do this in section 1.2.

Exercise 4 A sensible invariant on this state would be that a card cannot be both available and allocated at the same time. Extend the schema in Exercise 3 to include this invariant.

You now need to think about specifying the behaviour of the card issuing system in terms of its interface with the people who use it. You should consider possible operations, their inputs, outputs and their effect on the system state. For the purpose of these exercises we will define an operation to enter new cards into the system (thus making them available for allocation), and operations to allocate a card to an employee, return a card to the pool of available cards when an employee leaves and cancel a card which is reported as lost (making it unavailable for allocation).

Exercise 5 Using case analysis, define informally the four operations we have identified for the card issuing system. You will have to think about the responses that the system will give when an operation is attempted under various different conditions.

Exercise 6 Using the door control system specification to help you, try to formalise your definitions from Exercise 5. To do this you will have to think about the functions which may alter the state when an operation is successful. At this stage you do not have to worry about how the functions work, but you must name them and identify their arguments.

Exercise 7 Add schemas to show the formalised operations in Z notation.

The next step is to define the functions which have been identified in the system operations. Since the operations define the conditions under which each function can be invoked, you do not need to impose preconditions on the functions here.

Exercise 8 Define a function *makeAvailable* which adds cards to the pool of available cards in the system. The arguments to *makeAvailable* are the system state and a set of cards.

Exercise 9 Define a function to allocate a new card to an employee. This will mean that the card is no longer available for allocation.

Exercise 10 Define a function to remove a card which has been allocated and return it to the pool of available cards.

Exercise 11 Define a function to make a card unavailable (such as when it is reported lost). This function will make only a single card unavailable since it is most likely to be applied to one card at a time. Since a card which is lost may or may not have been allocated, this function will have to remove it from both the *Available* and *Allocated* components of the state.