

Z Specification Example

Computer System Version 2

Total schemas with error checking

1 Introduction

A computer system that has multiple domains: Admin, Purchase, Retail and Booking. It also has multiple departments: Estates, Sales, Management, Office. The following table defines which Department members can login to which Domain.

	Admin	Purchase	Retail	Booking
Estates	x	x		
Sales		x	x	x
Management	x	x	x	x
Office	x	x		

2 Types

[EMPLOYEE] set of all uniquely identifiable employees

Later we will need to respond to certain errors so here we need to define a RESPONSE type that is a freetype - a set of defined elements that make up the type.

RESPONSE ::= *LoggedIn* | *NotLoggedIn* | *NotAUser* | *AlreadyAUser* | *OK*

3 The State of the System

<i>Computer</i>	_____
<i>users, loggedIn</i> : $\mathbb{P} \text{ EMPLOYEE}$	
<i>loggedIn</i> $\subseteq$ <i>users</i>	

4 Initialisation

<i>InitComputer</i>	_____
<i>Computer'</i>	
<i>loggedIn'</i> = $\emptyset$	
<i>users'</i> = $\emptyset$	

5 Change State Operations

Total Operations defined here are:

- i) Add User
- ii) Remove User
- iii) Log in
- iv) Log out

5.1 Add User

Adds a new user. This is composed of two schema $AddUser_0$ and $AddUserError$. They will be combined into one using the schema disjunction operator ($\vee$). Why disjunction? Because either the $AddUser_0$ will be true resulting in a new user OR the $AddUserError$ will be true resulting in an error message being displayed.

$AddUser_0$ specifies that $p?$ is a member of staff and should not be a member of users before the operation. It also specifies that after the completion of $AddUser$ the set users' will include the new user, $p?$ Any implementation should also not allow a change to the $loggedIn$ set after the operation.

$AddUser_0$	_____
$\Delta Computer$	
$p? : EMPLOYEE$	
$p? \notin users$	
$users' = users \cup \{p?\}$	
$loggedIn' = loggedIn$	

$AddUserError$ completes the $AddUser$ operation by specifying how to deal with any errors. Note the use of the Ξ decoration which means there is no change to the state of the system by the $AddUserError$ schema, it just declares a $reply!$ output variable of type $RESPONSE$ and specifies the system outputs "AlreadyAUser" when $p?$ is indeed already a user.

$AddUserError$	_____
$\Xi Computer$	
$p? : EMPLOYEE$	
$reply! : RESPONSE$	
$p? \in users$	
$reply! = AlreadyAUser$	

Disjunction of $AddUser_0$ with $AddUserError$ almost gives a complete $AddUser$ schema but shouldn't the $AddUser$ schema also output an all OK response when a user is successfully added? Why not. Disjunction of the two schemas and the addition of an OK response to the $AddUser_0$ schema is done below using a linear schema definition but alternatively you could have instead drawn another schema representing the OK Response, $AddUser_0$ and the $AddUser$ together. The completed $AddUser$ schema is:

$$AddUser == (AddUser_0 \wedge [reply! : RESPONSE \mid reply! = OK]) \vee AddUserError$$

5.2 Remove User

Removes a user. The total remove User schema is described in a similar manner to the $AddUser$ schema in the previous section. We define a $RemoveUser_0$ schema first. It specifies that $p?$ is a member of staff and constrains $p?$ to be a

member of users before the operation. It also specifies that $p?$ will be removed from users'. Any implementation should also ensure that $p?$ is not logged in before the operation. Any implementation should also not allow a change to the `loggedIn` set after the operation.

<i>RemoveUser₀</i>	_____
$\Delta Computer$ $p? : EMPLOYEE$	
$p? \in users$ $users' = users \setminus \{p?\}$ $p? \notin loggedIn$ $loggedIn' = loggedIn$	

The `RemoveUserError` specifies that $p?$ should not be a user and a response "NotAUser" is specified as an output. Another error would occur if $p?$ is a user and logged in. We shouldn't remove a user while they are logged in so this schema specifies that where $p?$ is a user and is logged in the reply "LoggedIn" should be output. Note the disjunction $\vee$ (or) that allows us to deal with both the case when $p?$ is not a user and also when $p?$ is a user.

<i>RemoveUserError</i>	_____
$\Xi Computer$ $p? : EMPLOYEE$ $reply! : RESPONSE$	
$p? \notin users$ $reply! = NotAUser$ $\vee$ $p? \in users \wedge p? \in loggedIn \wedge response! = LoggedIn$	

$RemoveUser ==$
 $(RemoveUser_0 \wedge [reply! : RESPONSE \mid reply! = OK]) \vee RemoveUserError$

5.3 Log In

The total `logIn` specification is described in a similar manner to the `RemoveUser` schema in the previous section. We define a the `LogIn0` schema first. It specifies that $p?$ is a member of staff and should be a member of users before the operation. It also specifies that $p?$ should not be a member of `loggedIn` before the operation. It also specifies `loggedIn'` will include the new user, $p?$ after the operation.

$LogIn_0$ $\Delta Computer$ $p? : EMPLOYEE$
$p? \in users$ $p? \notin loggedIn$ $users' = users$ $loggedIn' = loggedIn \cup \{p?\}$

An error should be flagged up if someone who is not a user tries to login or when a user is already loggedin.

$LogInError$ $\Xi Computer$ $p? : EMPLOYEE$ $reply! : RESPONSE$
$(p? \notin users \wedge reply! = NotAUser)$ $\vee$ $(p? \in users \wedge p? \in loggedIn \wedge reply! = LoggedIn)$

Disjunction of ($LogIn_0$ and an OK response) with $LogInError$ now gives a complete $LogIn$ schema:

$LogIn == (LogIn_0 \wedge [reply! : RESPONSE \mid reply! = OK]) \vee LoginError$

5.4 Log Out

As before we will create the total LogOut schema by combining LogIn₀ and LoginError schemas using a schema disjunction. The basic log out specification is logOut₀. It specifies that $p?$ is a member of staff and should be a member of users before the operation. It also specifies that $p?$ should be a member of loggedIn before the operation. It also specifies that $p?$ will be removed from loggedIn' after the operation and that there will be no change in the set users' after the operation.

<i>LogOut₀</i>	_____
$\Delta Computer$	
$p? : EMPLOYEE$	
$p? \in users$	
$p? \in loggedIn$	
$users' = users$	
$loggedIn' = loggedIn \setminus \{p?\}$	

The LogoutError specifies that a LogOut error will have occurred if a person who is not a user tries to logout or if a user who is not logged in tries to logout and the system will output a suitable logout error response.

<i>LogoutError</i>	_____
$\Xi Computer$	
$p? : EMPLOYEE$	
$reply! : RESPONSE$	
$(p? \notin users \wedge reply! = NotAUser)$	
$\vee$	
$(p? \in users \wedge p? \notin loggedIn \wedge reply! = NotLoggedIn)$	

Disjunction of (LogOut₀ and an OK response) with LogoutError now gives a complete LogOut schema:

$$LogOut == (LogOut_0 \wedge [reply! : RESPONSE \mid reply! = OK]) \vee LogoutError$$