

Z Specification Example

Computer System Version 3

Extending schemas with relations

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 and Globals

[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* | *AlreadyInADepartment* | *NotInADepartment* | *CannotAccessDomain* | *OK*

We'll need to initialise some more free types for working with relations later so here we go. For section 5.3 and 5.4 we need responses:-

DOMAIN::= *Admin* | *Purchase* | *Retail* | *Booking*

DEPARTMENT::= *Estates* | *Sales* | *Management* | *Office*

The following relations describe the relationships between DEPARTMENT and DOMAIN. This relation defines which department can access which computer domains.

Axiomatic box declaration for global

| *canAccess* : *DEPARTMENT* $\leftrightarrow$ *DOMAIN*

3 The State of the System

Computer

$users : \mathbb{P} EMPLOYEE$

$belongsTo : EMPLOYEE \leftrightarrow DEPARTMENT$

$loggedIn : EMPLOYEE \leftrightarrow DOMAIN$

4 Initialisation

We'll initialise `canAccess` here but it should really be empty initially and then operations defined to add or remove maplets in accordance with the specification.

InitComputer

Computer'

$loggedIn' = \emptyset$

$users' = \emptyset$

$belongsTo' = \emptyset$

$canAccess' = \{ Estates \mapsto Admin, Estates \mapsto Purchase,$

$Sales \mapsto Purchase, Sales \mapsto Retail, Sales \mapsto Booking,$

$Management \mapsto Admin, Management \mapsto Purchase,$

$Management \mapsto Retail, Management \mapsto Booking,$

$Office \mapsto Admin, Office \mapsto Purchase \}$

5 Change State Operations

Total Operations defined here are:

- i) Add User
- ii) Remove User
- iii) Join Department
- iv) Leave Department
- v) Log in
- vi) Log out

5.1 Add User

Adds a new user. This is composed of two schema $AddUser_0$ and $AddUserError$.

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

$AddUserError$ completes the $AddUser$ operation by specifying how to deal with any errors.

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

Disjunction of the two schemas and the addition of an OK response to the $AddUser_0$ schema completes the $AddUser$ schema:

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

5.2 Remove User

Removes a user. $p?$ must be a user, must not belong to a department or be logged in. No other state changes to occur.

$RemoveUser_0$ $\Delta Computer$ $p? : EMPLOYEE$	
$p? \in users$ $users' = users \setminus \{p?\}$ $p? \notin \text{dom } loggedIn$ $p? \notin \text{dom } belongsTo$ $loggedIn' = loggedIn$ $belongsTo' = belongsTo$	

RemoveUserError $\Xi \text{Computer}$ $p? : \text{EMPLOYEE}$ $\text{reply!} : \text{RESPONSE}$
$p? \notin \text{users} \wedge \text{reply!} = \text{NotAUser}$ $\vee$ $p? \in \text{users} \wedge p? \in \text{dom belongsTo} \wedge \text{response!} = \text{AlreadyInADepartment}$ $\vee$ $p? \in \text{users} \wedge p? \in \text{loggedIn} \wedge \text{response!} = \text{LoggedIn}$

$\text{RemoveUser} ==$
 $(\text{RemoveUser}_0 \wedge [\text{reply!} : \text{RESPONSE} \mid \text{reply!} = \text{OK}]) \vee \text{RemoveUserError}$

5.3 Join Department

We define the JoinDept_0 schema first. It specifies that $p?$ will join department $d?$ but $p?$ must already be a user and not logged in. $p?$ cannot join a department if they are already a member of another department. We will use the belongsTo relation for this operation together with the use of dom and ran operators.

The relation belongsTo is really a function so we could use the overriding $\oplus$ operator to we-write a maplet.

JoinDept_0 $\Delta \text{Computer}$ $p? : \text{EMPLOYEE}$ $\text{dep?} : \text{DEPARTMENT}$
$p? \in \text{users}$ $p? \notin \text{dom belongsTo}$ $\text{users}' = \text{users}$ $\text{belongsTo}' = \text{belongsTo} \cup \{p? \mapsto \text{dep?}\}$

JoinDeptError $\Xi \text{Computer}$ $p? : \text{EMPLOYEE}$ $\text{dep?} : \text{DEPARTMENT}$ $\text{reply!} : \text{RESPONSE}$
$p? \notin \text{users} \wedge \text{reply!} = \text{NotAUser}$ $\vee$ $p? \in \text{users} \wedge p? \in \text{dom belongsTo} \wedge \text{reply!} = \text{AlreadyInADep}$

Disjunction of $(\text{JoinDept}_0$ and an OK response) with JoinDeptError now gives a complete JoinDept schema:

$\text{JoinDept} == (\text{JoinDept}_0 \wedge [\text{reply!} : \text{RESPONSE} \mid \text{reply!} = \text{OK}]) \vee \text{JoinDeptError}$

5.4 Leave Department

We define the LeaveDept_0 schema first. It specifies that $p?$ will leave department $d?$ but $p?$ must already be a user and not logged in. $p?$ cannot leave a department if they are not a member of a department. We will use the belongsTo relation for this operation together with the use of dom and ran operators. The relation belongsTo is really a function so we could use other function operators to remove a maplet instead.

LeaveDept_0
$\Delta \text{Computer}$ $p? : \text{EMPLOYEE}$ $dep? : \text{DEPARTMENT}$
$p? \in \text{users}$ $p? \in \text{dom } \text{belongsTo}$ $\text{users}' = \text{users}$ $\text{belongsTo}' = \text{belongsTo} \setminus \{p? \mapsto dep?\}$ $p? \notin \text{dom } \text{loggedIn}$ $\text{loggedIn}' = \text{loggedIn}$
LeaveDeptError
$\exists \text{Computer}$ $p? : \text{EMPLOYEE}$ $dep? : \text{DEPARTMENT}$ $\text{reply!} : \text{RESPONSE}$
$p? \notin \text{users} \wedge \text{reply!} = \text{NotAUser}$ $\vee$ $p? \in \text{users} \wedge p? \notin \text{dom } \text{belongsTo} \wedge \text{reply!} = \text{NotInADepartment}$ $\vee$ $p? \in \text{users} \wedge p? \in \text{dom } \text{loggedIn} \wedge \text{reply!} = \text{LoggedIn}$

Disjunction of (LeaveDept_0 and an OK response) with LeaveDeptError now gives a complete LeaveDept schema:
 $\text{LeaveDept} == (\text{LeaveDept}_0 \wedge [\text{reply!} : \text{RESPONSE} \mid \text{reply!} = \text{OK}]) \vee \text{LeaveDeptError}$

5.5 Log In

We define the LogIn_0 schema first. It specifies that $p?$ should be a member of users and has joined a department before the operation. It also specifies that $p?$ should not be logged in before the operation. We test this by checking whether the dom of loggedIn contains $p?$. It also specifies $\text{loggedIn}'$ will include the new maplet $p? \mapsto d?$ after the operation.

LogIn_0 $\Delta \text{Computer}$ $p? : \text{EMPLOYEE}$ $d? : \text{DOMAIN}$
$p? \in \text{users}$ $p? \notin \text{dom } \text{loggedIn}$ $\{p? \mapsto d?\} \in \text{belongsTo} \circ \text{canAccess}$ $\text{users}' = \text{users}$ $\text{belongsTo}' = \text{belongsTo}$ $\text{loggedIn}' = \text{loggedIn} \cup \{p? \mapsto d?\}$

An error should be flagged up if someone who is not a user tries to login or when a user is already logged in. We also check whether $p?$ has access rights to the computer domain, $d?$. If not then login fails.

LogInError $\Xi \text{Computer}$ $p? : \text{EMPLOYEE}$ $d? : \text{DOMAIN}$ $\text{reply!} : \text{RESPONSE}$
$p? \notin \text{users} \wedge \text{reply!} = \text{NotAUser}$ $\vee$ $p? \in \text{users} \wedge \{p? \mapsto d?\} \notin \text{belongsTo} \circ \text{canAccess} \wedge \text{reply!} = \text{CannotAccessDomain}$ $\vee$ $p? \in \text{users} \wedge p? \in \text{dom } \text{loggedIn} \wedge \text{reply!} = \text{LoggedIn}$

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

$\text{LogIn} == (\text{LogIn}_0 \wedge [\text{reply!} : \text{RESPONSE} \mid \text{reply!} = \text{OK}]) \vee \text{LogInError}$

5.6 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 $p?$ should be a member of users and belong to a department before the operation. It also specifies that $p?$ should be logged in before the operation. It also specifies that the $\{p? \mapsto d?\}$ maplet will be removed from the loggedIn' relation after the operation and that there will be no other change to the computer system's state after the operation.

<i>LogOut₀</i>
$\Delta Computer$
$p? : EMPLOYEE$
$d? : DOMAIN$
$p? \in users$
$\{p? \mapsto d?\} \in loggedIn$
$p? \in \text{dom } belongsTo$
$users' = users$
$belongsTo' = belongsTo$
$loggedIn' = loggedIn \setminus \{p? \mapsto d?\}$

The LogoutError specifies that a Log out error has occurred if a person who is not a user tries to log out or if a user who is not logged in tries to log out or if a person who is not a member of a department tries to log out and the system outputs a suitable error response.

<i>LogoutError</i>
$\Xi Computer$
$p? : EMPLOYEE$
$d? : DOMAIN$
$reply! : RESPONSE$
$p? \notin users \wedge reply! = NotAUser$
$\vee$
$p? \in users \wedge \{p? \mapsto d?\} \notin loggedIn \wedge reply! = NotLoggedIn$
$\vee$
$p? \in users \wedge p? \notin \text{dom } belongsTo \wedge reply! = NotInADepartment$

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

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