

# Versions Control System

Group 9

Jeffrey Grela 6734326

Shawn Leslie 6721988

Amy Lequier 6705573

A number is any digit 0..9.

$$\textit{Number} ::= \{n : \mathbb{Z} \mid 0 \leq n \leq 9\}$$

A character is an individual character, which is useful for character specific attributes such as userids and passwords.

$$[\textit{CHAR}]$$

A string is a sequence of characters unconstrained by length.

$$[\textit{STRING}]$$

The free type of *UserStatus* is used by the *User* schema to signify the status of the user.

$$\textit{UserStatus} ::= \textit{Active} \mid \textit{Inactive}$$

Typical boolean definitions.

$$\textit{Boolean} ::= \textit{True} \mid \textit{False}$$

A maximum is some natural number. This is used in calculating limits for passwords and userids.

$$\mid \textit{Maximum} : \mathbb{N}_1$$

There are two user types, *Manager* and *Designer*. The *manager* user type will have predominantly administrative abilities, whereas *designers* will be responsible for inputting data to the system.

$$\textit{UserType} ::= \textit{Manager} \mid \textit{Designer}$$

A user has a name, which is a sequence of characters, a group membership identifier, a password used to authenticate a user to the system, an employee number, and a status that determines whether or not the user is using the system. Password, employee number and name must be less than or equal to 8 characters.

*User*

$name : \text{seq}_1 \text{ CHAR}$   
 $group : \text{UserType}$   
 $employeeNumber : \text{seq}_1 \text{ Number}$   
 $password : \text{seq}_1 \text{ CHAR}$   
 $status : \text{UserStatus}$

$\#name \leq \text{Maximum}$   
 $\#password \leq \text{Maximum}$   
 $\#employeeNumber \leq \text{Maximum}$

All users are contained in this schema. In the relation defined by employee numbers and names, there can only be one employee number for each name, and there can only be only name for each employee number. Additionally, there can be no duplicate users – if either the name or the employee number are the same, then the two users are the same user.

*AllUsers*

$users : \mathbb{P} \text{ User}$   
 $relation : \text{seq}_1 \text{ CHAR} \mapsto \text{seq}_1 \text{ Number}$

$\forall x : \text{User}; y : \text{seq}_1 \text{ CHAR} \mid$   
 $x \in users \wedge y \in \text{dom } relation \bullet$   
 $x.name = y \Leftrightarrow$   
 $(\exists_1 z : \text{User} \mid z \in users \bullet z.name = y)$   
 $\forall x : \text{User}; y : \text{seq}_1 \text{ Number} \mid$   
 $x \in users \wedge y \in \text{ran } relation \bullet$   
 $x.employeeNumber = y \Leftrightarrow$   
 $(\exists_1 z : \text{User} \mid z \in users \bullet z.employeeNumber = y)$   
 $\forall x, y : \text{User} \mid x \in users \wedge y \in users \bullet$   
 $x = y \Leftrightarrow$   
 $(x.name = y.name \vee$   
 $x.employeeNumber = y.employeeNumber)$

There should be no users or relationships when the system initializes.

*InitAllUsers*

$users : \mathbb{P} \text{ User}$   
 $relation : \text{seq}_1 \text{ CHAR} \mapsto \text{seq}_1 \text{ Number}$

$users = \emptyset$   
 $relation = \emptyset$

A user, with a password and a userid, is added to the set of all users if no previous user shares that same ID or that same employee number. All information is assumed to be

inputted, including group membership. We also create a new employee number and user name relationship. (VC01)

$\textit{Register}$ <hr/> $\Delta \textit{AllUsers}$ $\textit{newuser?} : \textit{User}$ <hr/> $\forall x : \textit{User} \mid x \in \textit{users} \bullet$ $x.\textit{name} \neq \textit{newuser?}.\textit{name} \wedge$ $x.\textit{employeeNumber} \neq \textit{newuser?}.\textit{employeeNumber}$ $\textit{users}' = \textit{users} \cup \{\textit{newuser?}\}$ $\textit{relation}' = \textit{relation} \oplus$ $\{(\textit{newuser?}.\textit{name} \mapsto \textit{newuser?}.\textit{employeeNumber})\}$
---

Given a userID, it is possible to delete one user from the system if that user exists already. We must also remove the relation that holds between username and employee number. (VC13)

$\textit{UnRegister}$ <hr/> $\Delta \textit{AllUsers}$ $\textit{username?} : \text{seq}_1 \textit{CHAR}$ <hr/> $\exists_1 x : \textit{User} \mid x \in \textit{users} \bullet x.\textit{name} = \textit{username?}$ $\textit{users}' = \textit{users} \setminus$ $\{a : \textit{User} \mid a \in \textit{users} \wedge a.\textit{name} = \textit{username?} \bullet a\}$ $\textit{relation}' = \{\textit{username?}\} \triangleleft \textit{relation}$
--

We can change the Role of a User, if that user exists. (VC14)

$\textit{ChangeRole}$ <hr/> $\Delta \textit{AllUsers}$ $\textit{username?} : \text{seq}_1 \textit{CHAR}$ $\textit{newRole?} : \textit{UserType}$ <hr/> $\exists_1 x : \textit{User} \mid x \in \textit{users}' \wedge x.\textit{name} = \textit{username?} \bullet x.\textit{group} = \textit{newRole?}$
--

Changing the password can be done as long as the new password is not the same as the old password. If this pre-condition is satisfied, then we can change the password.

<p><i>ChangePassword</i></p> <p><math>\Delta AllUsers</math></p> <p><math>username? : seq_1 CHAR</math></p> <p><math>newPassword? : seq_1 CHAR</math></p> <hr/> <p><math>\exists_1 x : User \mid x \in users \wedge x.name = username? \bullet</math>  <math>x.password \neq newPassword?</math></p> <p><math>\exists_1 x : User \mid x \in users' \wedge x.name = username? \bullet</math>  <math>x.password = newPassword?</math></p>
---

Logging in to the system with a valid ID and a correct password will change a user's status from Inactive to Active. (VC02)

<p><i>Login</i></p> <p><math>\Delta AllUsers</math></p> <p><math>username? : seq_1 CHAR</math></p> <p><math>password? : seq_1 CHAR</math></p> <hr/> <p><math>\exists_1 x : User \mid x \in users \bullet</math>  <math>x.name = username? \wedge x.status \neq Active \wedge</math>  <math>x.password = password?</math></p> <p><math>\exists_1 x : User \mid x \in users' \wedge x.name = username? \bullet</math>  <math>x.status = Active</math></p>
---

Logging out of the system with a valid ID will change a user's status from Inactive to Active. (VC02)

<p><i>Logout</i></p> <p><math>\Delta AllUsers</math></p> <p><math>username? : seq_1 CHAR</math></p> <hr/> <p><math>\exists_1 x : User \mid x \in users \bullet</math>  <math>x.name = username? \wedge x.status \neq Inactive</math></p> <p><math>\exists_1 x : User \mid x \in users' \wedge x.name = username? \bullet</math>  <math>x.status = Inactive</math></p>
---

This function will return a boolean value if the user specified by the sequence of characters representing their name has a valid attribute. (VC02)

<p><math>IsValid : seq_1 CHAR \rightarrow Boolean</math></p> <p><math>\exists AllUsers</math></p> <hr/> <p><math>\forall x : seq_1 CHAR; y : Boolean \bullet</math>  <math>(\forall z : User \mid z \in users \bullet</math>  <math>(z.name = x \wedge z.status = Active \Leftrightarrow IsValid(x) = True) \vee</math>  <math>IsValid(x) = False)</math></p>
---

A Co-ordinate consists of two integers which determine a location.

$$Coord == \mathbb{Z} \times \mathbb{Z}$$

Returns the first element (X) in a tuple which represents a coordinate system.

$$\frac{X : Coord \rightarrow \mathbb{Z}}{\forall x, y : \mathbb{Z} \mid (x, y) \in Coord \bullet X(x, y) = x}$$

Returns the second element (Y) in a tuple which represents a coordinate system.

$$\frac{Y : Coord \rightarrow \mathbb{Z}}{\forall x, y : \mathbb{Z} \mid (x, y) \in Coord \bullet Y(x, y) = y}$$

There are several object types available.

$$ObjectType ::= Door \mid IntWall \mid ExtWall \mid Bathtub \mid Sink$$

A unique identifier for an Object.

$$[OBJECTID]$$

The object schema, which consists of two co-ordinates, the unique identifier for the object, and the type of the object. The start version and end versions represent the lifespan of the object: The start version is the version number that the object first appeared in, whereas the the endversion is the version number where that object was deleted.

$$\frac{object}{\begin{array}{l} A : Coord \\ B : Coord \\ name : OBJECTID \\ type : ObjectType \\ versions : \mathbb{P} \mathbb{Z} \end{array}}$$

A unique identifier for an enclosure.

$$[ENCLOSUREID]$$

There are two types of enclosures: interior and exterior.

$$EnclosureType ::= Interior \mid Exterior$$

An enclosure, or a room, consists of several objects. Each of these objects must have a unique ID. The enclosures also have a pair of co-ordinate tuples to identify their position on the layout of the house plan.

*enclosure*

$allObjects : \mathbb{F} \textit{object}$   
 $name : ENCLOSUREID$   
 $type : EnclosureType$   
 $owner : STRING$

A project consists of several enclosures, or rooms. Each room is owned by a specific user, as is each object. There can only be one user who owns each object, as is signified by the parietal function between the two relationships. Also, each object has a relationship with one specific enclosure.

*project*

$allEnclosures : \mathbb{F} \textit{enclosure}$   
 $allObjects : \mathbb{F} \textit{object}$   
 $projectName : STRING$   
 $ownsObject : \textit{object} \rightarrow \text{seq}_1 \textit{CHAR}$   
 $room : \textit{object} \rightarrow \textit{enclosure}$   
 $currentVersion : \mathbb{Z}$

$\forall x, y : \textit{enclosure} \mid x \in allEnclosures \wedge y \in allEnclosures \bullet$   
 $x = y \Leftrightarrow x.name = y.name$   
 $\forall x, y : \textit{object} \mid x \in allObjects \wedge y \in allObjects \bullet$   
 $x = y \Leftrightarrow x.name = y.name$

Everything should be empty at the beginning of a new project. We query for the name of the project. The first version of a project is set to be '0'. (VC07)

*initProject*

$\Delta \textit{project}$   
 $name? : STRING$   
 $allEnclosures = \emptyset$   
 $allObjects = \emptyset$   
 $ownsObject = \emptyset$   
 $room = \emptyset$   
 $projectName' = name?$   
 $currentVersion' = 0$

A session is a time period where one user works on the system and adds and deletes objects and enclosures from the system. Each session is attached to one user in the system. Each session also has a start time and an ending time, where the end time must be later than the starting time. There must be no one object or one enclosure in their respective addition and deletion sets. The valid flag checks to see if the session's user intentions have

been verified by the system. Current objects and current enclosures represent the state of the project as it was when the user initially checked out the system. The current version is the version number of the project that the user is basing their intentions upon.

<i>session</i>
<i>addObjects</i> : $\mathbb{F}$ <i>object</i>
<i>delObjects</i> : $\mathbb{F}$ <i>object</i>
<i>addEnclosures</i> : $\mathbb{F}$ <i>enclosure</i>
<i>delEnclosures</i> : $\mathbb{F}$ <i>enclosure</i>
<i>currentObjects</i> : $\mathbb{F}$ <i>object</i>
<i>currentEnclosures</i> : $\mathbb{F}$ <i>enclosure</i>
<i>sessionVersion</i> : $\mathbb{Z}$
<i>name</i> : $\text{seq}_1$ <i>CHAR</i>
<i>timeStart</i> : $\mathbb{Z}$
<i>timeEnd</i> : $\mathbb{Z}$
<i>active</i> : <i>Boolean</i>
<i>valid</i> : <i>Boolean</i>
<i>objectOfEnclosure</i> : <i>object</i> $\rightarrow$ <i>enclosure</i>
<i>addObjects</i> $\cap$ <i>delObjects</i> = $\emptyset$
<i>addEnclosures</i> $\cap$ <i>delEnclosures</i> = $\emptyset$
<i>timeStart</i> < <i>timeEnd</i>

This is defined to be the current system time.

| *CurrentTime* :  $\mathbb{N}_1$

Each session initialization must have a valid login to be created: This is done by inheriting the Login schema. If the preconditions for this are met, then we know that a valid user is using the system. We can then initialize each set of added and deleted objects and enclosures to be empty, and we can assign the time that this session was initialized to be the current time. The active flag is also set for this session. As this session has not been checked and validated, the valid flag is set to false. (VC03)

<i>initSession</i>
$\exists \text{Login}$
$\exists \text{project}$
$\text{newSession}' : \text{session}$
$\text{newSession}'.\text{name} = \text{username?}$
$\text{newSession}'.\text{timeStart} = \text{currentTime}$
$\text{newSession}'.\text{active} = \text{True}$
$\text{newSession}'.\text{objectOfEnclosure} = \text{room}$
$\text{newSession}'.\text{currentObjects} = \text{allObjects}$
$\text{newSession}'.\text{currentEnclosures} = \text{allEnclosures}$
$\text{newSession}'.\text{addObjects} = \emptyset$
$\text{newSession}'.\text{delObjects} = \emptyset$
$\text{newSession}'.\text{addEnclosures} = \emptyset$
$\text{newSession}'.\text{delEnclosures} = \emptyset$
$\text{newSession}'.\text{valid} = \text{False}$
$\text{newSession}'.\text{sessionVersion} = \text{currentVersion}$

The de-initialization of a session is simply the input of a session to be deleted at the current time. (VC04)

<i>deInitSession</i>
$\exists \text{Logout}$
$\text{delSession}, \text{delSession}' : \text{session}$
$\text{username?} = \text{delSession}.name$
$\text{delSession}'.\text{timeEnd} = \text{currentTime}$
$\text{delSession}'.\text{active} = \text{False}$

A user cannot have more than two sessions open at the same time: if two sessions of the same user are present, then they are the same session.

<i>allSessions</i>
$\exists \text{project}$
$\text{allSessions} : \mathbb{P} \text{session}$
$\forall x, y : \text{session} \mid x \in \text{allSessions} \wedge y \in \text{allSessions} \bullet$
$\quad x = y \Leftrightarrow x.name = y.name$
$\forall x : \text{session} \mid x \in \text{allSessions} \bullet x.sessionVersion \leq \text{currentVersion}$

There are no sessions open at the start of the system.

<i>initAllSessions</i>
$\text{sessions} : \mathbb{P} \text{session}$
$\text{sessions} = \emptyset$

If the pre-conditions are met by the `initSession` schema, then we can add this session to the set of all open sessions.

$\frac{\text{addSession}}{\Delta allSessions \quad \exists initSession}$
$allSessions' = allSessions \cup \{newSession'\}$

If the pre and post conditions are met for deletion of a session, we can delete the session from the set of all sets.

$\frac{delSession}{\Delta allSessions \quad delSession? : session}$
$allSessions' = allSessions \setminus \{delSession?\}$

Adding a new object to the set of all objects. If the object is not already part of the set of all added objects, then we add this object to the set of all objects. Since all IDs for each object are unique, we need not worry about adding an object which has been previously deleted: the new object will be just that: a new object. (VC10)

$\frac{addObject}{\Delta session \quad newObject? : object \quad newEnclosure? : enclosure}$
$\forall x : object \mid x \in addObjects \bullet x \neq newObject?$
$addObjects' = addObjects \cup \{newObject?\}$
$objectOfEnclosure' = objectOfEnclosure \oplus \{newObject? \mapsto newEnclosure?\}$

Adding a new enclosure to the set of all enclosures. If the enclosure is not already part of the set of all added enclosures, then we add this enclosure to the set of all enclosure. The new enclosure cannot have any objects attached to it initially.

$\frac{addEnclosure}{\Delta session \quad newEnclosure? : enclosure}$
$newEnclosure?.allObjects = \emptyset$
$\forall x : enclosure \mid x \in addEnclosures \bullet x \neq newEnclosure?$
$addEnclosures' = addEnclosures \cup \{newEnclosure?\}$

Deleting objects and Enclosures are similar schemas. For an object to be properly deleted, the object must be present within either in the set of objects within the current project, or it must be present within the current set of added objects within this schema. We also delete all relationships which contain the object within the domain of that relationship. (VC11)

$\begin{array}{l} \underline{\text{delObject}} \\ \Delta_{\text{session}} \\ \text{delObject?} : \text{object} \\ \hline \forall x : \text{object} \mid x \in \text{delObjects} \bullet x \neq \text{delObject?} \\ \quad (\forall x : \text{object} \mid \\ \quad \quad x \in \text{currentObjects} \bullet x = \text{delObject?} \wedge \\ \quad \quad \text{delObjects}' = \text{delObjects} \cup \{\text{delObject?}\}) \vee \\ (\forall y : \text{object} \mid y \in \text{addObjects} \bullet y = \text{delObject?} \wedge \\ \quad \# \text{addObjects} \geq 1 \wedge \\ \quad \text{addObjects}' = \text{addObjects} \setminus \{\text{delObject?}\}) \\ \text{objectOfEnclosure}' = \{\text{delObject?}\} \triangleleft \text{objectOfEnclosure} \end{array}$
--

Deleting objects and Enclosures are similar schemas. For an enclosure to be properly deleted, the enclosure must be present within either in the set of enclosures within the current project, or it must be present within the current set of added enclosures within this schema. We also erase all objects contained within this enclosure. There must be more than one enclosure available to be deleted in the set of all enclosures that the user received as being current. We also erase all relationships which deal with this enclosure.

$\begin{array}{l} \underline{\text{delEnclosure}} \\ \Delta_{\text{session}} \\ \text{delEnclosure?} : \text{enclosure} \\ \hline \forall x : \text{enclosure} \mid x \in \text{delEnclosures} \bullet x \neq \text{delEnclosure?} \\ (\forall x : \text{enclosure} \mid x \in \text{currentEnclosures} \bullet \\ \quad x = \text{delEnclosure?} \wedge \\ \quad \text{delEnclosures}' = \text{delEnclosures} \cup \\ \quad \quad \{\text{delEnclosure?}\}) \vee \\ (\forall y : \text{enclosure} \mid y \in \text{delEnclosures} \bullet \\ \quad y = \text{delEnclosure?} \wedge \\ \quad \# \text{addEnclosures} \geq 1 \wedge \\ \quad \text{addEnclosures}' = \text{addEnclosures} \setminus \\ \quad \quad \{\text{delEnclosure?}\}) \\ \forall x : \text{object} \mid \\ \quad x \in \text{dom} \\ \quad (\text{objectOfEnclosure} \triangleright \{\text{delEnclosure?}\}) \bullet \\ \quad \text{delObjects}' = \text{delObjects} \cup \{x\} \\ \text{objectOfEnclosure}' = \text{objectOfEnclosure} \triangleright \{\text{delEnclosure?}\} \end{array}$
---

$d$  is the denominator within the line crossing finding algorithm.

$$\begin{array}{|l} \hline d : \text{object} \times \text{object} \rightarrow \mathbb{Z} \\ \hline \forall x, y : \text{object} \bullet \\ d(x, y) = (((X(y.A) - X(x.A)) * (Y(y.B) - Y(y.A))) - \\ ((Y(y.A) - Y(x.A)) * (X(y.B) - X(y.A)))) \end{array}$$

$t0$  is the first intersection point in the parametric form of the two lines. It is the point on the first line where the two lines intersect.

$$\begin{array}{|l} \hline t0 : \text{object} \times \text{object} \rightarrow \mathbb{Z} \\ \hline \forall x, y : \text{object} \bullet \\ t0(x, y) = d(x, y) \text{div} \\ (((X(x.B) - X(x.A)) * \\ (Y(y.B) - Y(y.A))) - \\ ((Y(x.B) - Y(x.A)) * \\ (X(y.B) - X(y.A)))) \end{array}$$

$u0$  is the second intersection point in the parametric form of the two lines. It is the point on the second line where the two lines intersect.

$$\begin{array}{|l} \hline u0 : \text{object} \times \text{object} \rightarrow \mathbb{Z} \\ \hline \forall x, y : \text{object} \bullet \\ u0(x, y) = (X(x.A) + (X(x.B) - X(x.A)) * t0(x, y) - \\ X(y.A)) \text{div} (X(y.B) - X(y.A)) \end{array}$$

$tc$  is the point on the second line where the first point on the second line intersects. This equation is used for lines that are co-incident.

$$\begin{array}{|l} \hline tc : \text{object} \times \text{object} \rightarrow \mathbb{Z} \\ \hline \forall x, y : \text{object} \bullet \\ tc(x, y) = ((X(y.A) - X(y.B)) \text{div} (X(x.B) - X(x.A))) \end{array}$$

$tc$  is the point on the second line where the second point on the second line intersects. This equation is used for lines that are co-incident.

$$\begin{array}{|l} \hline td : \text{object} \times \text{object} \rightarrow \mathbb{Z} \\ \hline \forall x, y : \text{object} \bullet \\ td(x, y) = ((X(y.A) - X(x.A)) \text{div} (X(x.B) - X(x.A))) \end{array}$$

Two lines are coincident if the denominator is 0 and if either  $tc$  or  $td$  (the end points on the second line) are on the first line. If both are greater than one or both are less than 0, then the two lines do not overlap.

$coincident : object \times object \rightarrow Boolean$

$\forall x, y : object \bullet$   
 $(d(x, y) \neq 0 \Leftrightarrow coincident(x, y) = False) \vee$   
 $(d(x, y) = 0 \wedge (tc(x, y) < 0 \wedge td(x, y) < 0) \vee$   
 $(tc(x, y) > 1 \wedge td(x, y) > 1) \Leftrightarrow$   
 $coincident(x, y) = False) \vee$   
 $(d(x, y) = 0 \wedge \neg (tc(x, y) < 0 \wedge$   
 $td(x, y) < 0) \vee$   
 $(tc(x, y) > 1 \wedge td(x, y) > 1) \Leftrightarrow$   
 $coincident(x, y) = True)$

A line overlaps another line if it is coincident or if the u0 and t0 points are both less than or equal to one and greater than or equal to 0.

$overlap : object \times object \rightarrow Boolean$

$\forall x, y : object \bullet$   
 $(d(x, y) \neq 0 \wedge$   
 $(t0(x, y) < 0 \vee t0(x, y) > 1) \Leftrightarrow$   
 $overlap(x, y) = False) \vee$   
 $(d(x, y) \neq 0 \wedge t0(x, y) \geq 0 \wedge$   
 $t0(x, y) \leq 1 \wedge u0(x, y) \geq 0 \wedge$   
 $u0(x, y) \leq 1 \Leftrightarrow$   
 $overlap(x, y) = True) \vee$   
 $overlap(x, y) = coincident(x, y)$

Connected objects share a set of points in common with each other.

$connect : object \times object \rightarrow Boolean$

$\forall x, y : object \bullet$   
 $((x.A = y.B \vee x.B = y.A \vee x.A = y.A \vee y.B = x.B) \wedge x$   
 $\neq y \Leftrightarrow connect(x, y) = True) \vee$   
 $\neg ((x.A = y.B \vee x.B = y.A \vee x.A = y.A \vee y.B = x.B) \Leftrightarrow$   
 $connect(x, y) = False) \vee$   
 $x = y \Leftrightarrow connect(x, y) = False$

An enclosed area is one where all objects that are of type wall each have two or more unique connected elements.

$$\overline{enclosed : \mathbb{F}_1 \text{ object} \rightarrow \text{Boolean}}$$

$$\begin{aligned} & \forall \text{allobjects} : \mathbb{F}_1 \text{ object}; x, y, z : \text{object} \mid \\ & (x.type = \text{IntWall} \vee x.type = \text{ExtWall}) \\ & \wedge (y.type = \text{IntWall} \vee y.type = \text{ExtWall}) \wedge \\ & (z.type = \text{IntWall} \vee z.type = \text{ExtWall}) \bullet \\ & ((\text{connect}(x, y) = \text{True} \wedge \\ & \text{connect}(x, z) = \text{True} \wedge y \neq z) \Leftrightarrow \\ & \text{enclosed}(\text{allobjects}) = \text{True}) \vee \\ & (\neg (\text{connect}(x, y) = \text{True} \wedge \\ & \text{connect}(x, z) = \text{True} \wedge y \neq z) \Leftrightarrow \\ & \text{enclosed}(\text{allobjects}) = \text{False}) \end{aligned}$$

This function returns true if a number of conditions are held by the set of objects given to it.

$$\overline{validPlan : \mathbb{F} \text{ object} \times \mathbb{F} \text{ enclosure} \rightarrow \text{Boolean}}$$

$$\begin{aligned} & \forall x : \mathbb{F} \text{ object}; y : \mathbb{F} \text{ enclosure} \bullet \\ & (\forall a, b : \text{object}; c : \text{enclosure} \mid \\ & a \in x \wedge b \in x \wedge c \in y \bullet \\ & \text{enclosed}(c.allObjects) = \text{True} \wedge \\ & \text{overlap}(a, b) = \text{False} \wedge \\ & a.type = \text{Door} \wedge \\ & ((b.type = \text{ExtWall}) \vee \\ & (b.type = \text{IntWall})) \\ & \Rightarrow \text{coincident}(a, b) = \text{True}) \\ & \Leftrightarrow \text{validPlan}(x, y) = \text{True} \vee \\ & (\text{validPlan}(x, y) = \text{False}) \end{aligned}$$

Verification is the testing of the effect that a set of user's intentions will have on the final project given the state of the project that the user checked in. If the addition of the user's intentions creates a valid system, then the valid flag is set to true, and the session is considered to be verified. (VC04)

$$\overline{\text{verification}}$$

$$\Delta \text{session}$$

$$\text{intAllObjects} : \mathbb{F} \text{ object}$$

$$\text{intAllEnclosures} : \mathbb{F} \text{ enclosure}$$

$$\text{intAllObjects} = \text{currentObjects} \cup \text{addObjects} \setminus \text{delObjects}$$

$$\text{intAllEnclosures} = \text{currentEnclosures} \cup \text{addEnclosures} \setminus \text{delEnclosures}$$

$$\text{valid}' = \text{validPlan}(\text{intAllObjects}, \text{intAllEnclosures})$$

This is the value that the system will assign to each permutation of the intentions and the current project.

|  $Value : \mathbb{Z}$

This is a function which describes the input and output of the semantical value function: Upon receiving a set of objects, this function will attempt to assign some value to this project.

|  $semanticValue : \mathbb{F} object \rightarrow \mathbb{Z}$   
 |  $\forall x : \mathbb{F} object \bullet semanticValue(x) = Value$

If two sessions happen to carry equal weight in either the correctness or in creating a maximal semantical value to the project, then we must choose one session over the other session. We do this by first seeing which version number the two sessions have: The session which is based upon the more recent version is the session that will be used to resolve ties. If the two versions are the same, then we consider the session whose check-in time is more recent to be the victor. (VC04)

|  $contention : session \times session \rightarrow session$   
 |  $\forall x, y : session \bullet$   
 |  $(x.sessionVersion > y.sessionVersion \Leftrightarrow$   
 |  $contention(x, y) = x) \vee$   
 |  $(y.sessionVersion > x.sessionVersion \Leftrightarrow$   
 |  $contention(x, y) = y) \vee$   
 |  $(y.sessionVersion = x.sessionVersion \wedge$   
 |  $y.timeEnd \geq x.timeEnd \Leftrightarrow$   
 |  $contention(x, y) = y) \vee$   
 |  $(y.sessionVersion = x.sessionVersion \wedge$   
 |  $x.timeEnd \geq y.timeEnd \Leftrightarrow$   
 |  $contention(x, y) = x)$

A merge is done for all sessions that satisfy the timing criteria: any user who checks in a set of intentions within the time period that the one user was modifying the data is considered to have created additional sets of intentions that are in contention with the original user's set of intentions. Some preconditions must be held before this schema is allowed to operate: A valid logout must be performed, and the current session must have been previously verified by the verification schema. The output of the schema must be a new set of allObjects within the project schema that is correct and has a maximal semantical value. A maximal semantical value is defined to the permutation of the intentions that will produce the highest value: thus, the output must have a higher value than all other possible permutations of the addition of intentions. All considerations are done at the object level: enclosures are used to provide a better description of the project. All added objects must then have added to them the new current version to signify their membership in the new project version. Deleted objects will not receive this designation, and will not be valid for this new version. (VC04)

*merge*

$\exists$ Logout

$\exists$ session

$\Delta$ project

Sessions :  $\mathbb{P}$  session

validSessions' :  $\mathbb{F}$  session

addedObjects' :  $\mathbb{F}$  object

delObjects' :  $\mathbb{F}$  object

allPossibleObjects' :  $\mathbb{F}$  object

allPossibleDelObjects' :  $\mathbb{F}$  object

*valid* = True

*name* = username?

$\forall x : \text{session} \mid x \in \text{Sessions} \wedge x.\text{valid} = \text{True} \bullet$   
     $((x.\text{timeEnd} < \text{timeEnd} \wedge x.\text{timeEnd} > \text{timeStart})$   
     $\vee x.\text{name} = \text{name}) \wedge$   
     $\text{validSessions}' = \text{validSessions}' \cup \{x\} \wedge$   
     $\text{allPossibleObjects}' = \text{allPossibleObjects}'$   
     $\cup x.\text{addObjects}$

$\forall x : \text{session} \mid x \in \text{Sessions} \wedge$   
     $x.\text{valid} = \text{True} \bullet$   
     $x.\text{timeEnd} < \text{timeEnd} \wedge$   
     $x.\text{timeEnd} > \text{timeStart} \wedge$   
     $\text{allPossibleDelObjects}' = \text{allPossibleDelObjects}' \cup$   
     $x.\text{delObjects}$

$\text{addedObjects}' \subseteq \text{allPossibleObjects}'$   
 $\text{delObjects}' \subseteq \text{allPossibleDelObjects}'$   
 $\text{validPlan}(\text{allObjects}', \text{allEnclosures}') = \text{True}$   
 $\text{currentVersion}' = \text{currentVersion} + 1$

$\forall x, y : \mathbb{F} \text{ object} \mid$   
     $x \subseteq \text{allPossibleObjects}' \wedge$   
     $y \subseteq \text{allPossibleDelObjects}' \bullet$   
     $(\text{semanticValue}(\text{allObjects}' \cup \text{addedObjects}') \geq$   
     $\text{semanticValue}(\text{allObjects}' \cup x \setminus y)) \wedge$   
     $\text{validPlan}(\text{allObjects}' \cup \text{addedObjects}'$   
     $\setminus \text{delObjects}', \text{allEnclosures}') = \text{True}$

$\forall x : \text{object} \mid x \in \text{addedObjects}' \bullet$   
     $x.\text{versions} = x.\text{versions} \cup \{\text{currentVersion}\}$

$\forall x, y : \text{object} \mid x \in \text{addObjects} \wedge$   
     $y \in \text{allPossibleObjects}' \bullet$   
     $x = y \Rightarrow (\text{ownsObject}' = \text{ownsObject} \oplus$   
     $\{(x \mapsto \text{name})\})$