Julien (GUÉPIN | VAUBOURG)
✉ julien@(guepin.fr | vaubourg.com)
2A groupe GG1

Gestionnaire d'agendas multiples

Conception

CESI

Le 9 novembre 2011

Table des matières

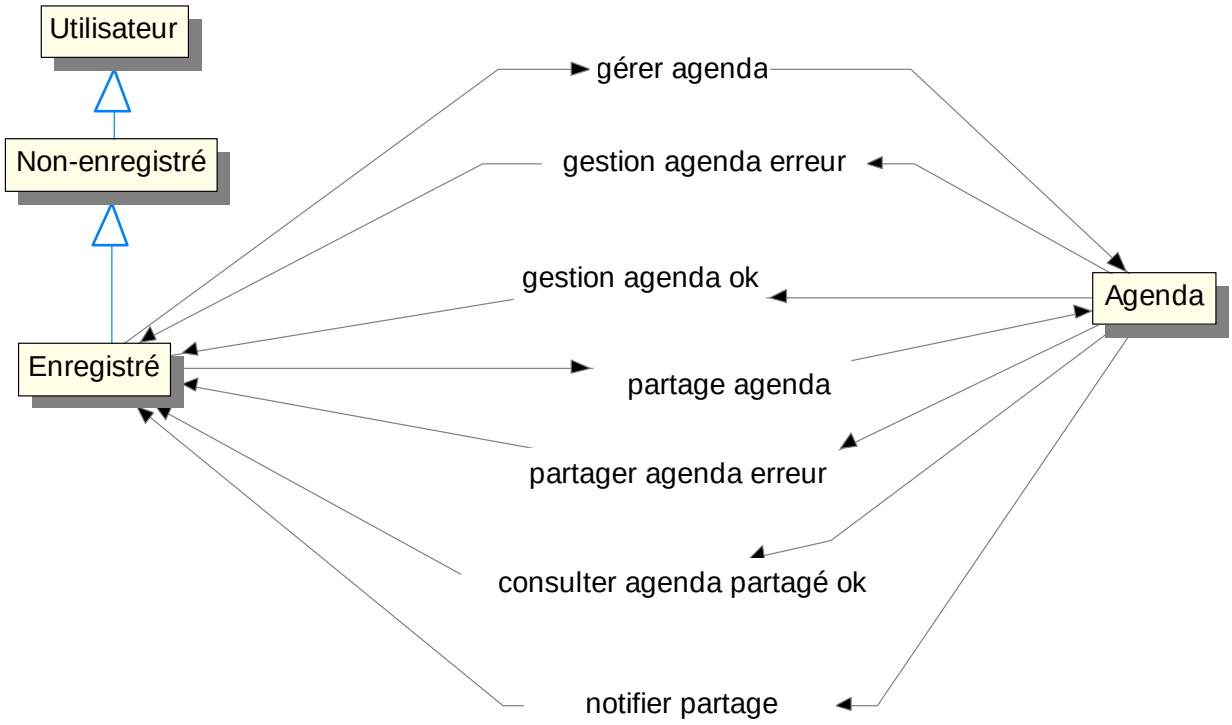
1 Modèles conceptuels

1.1 Diagrammes de flux

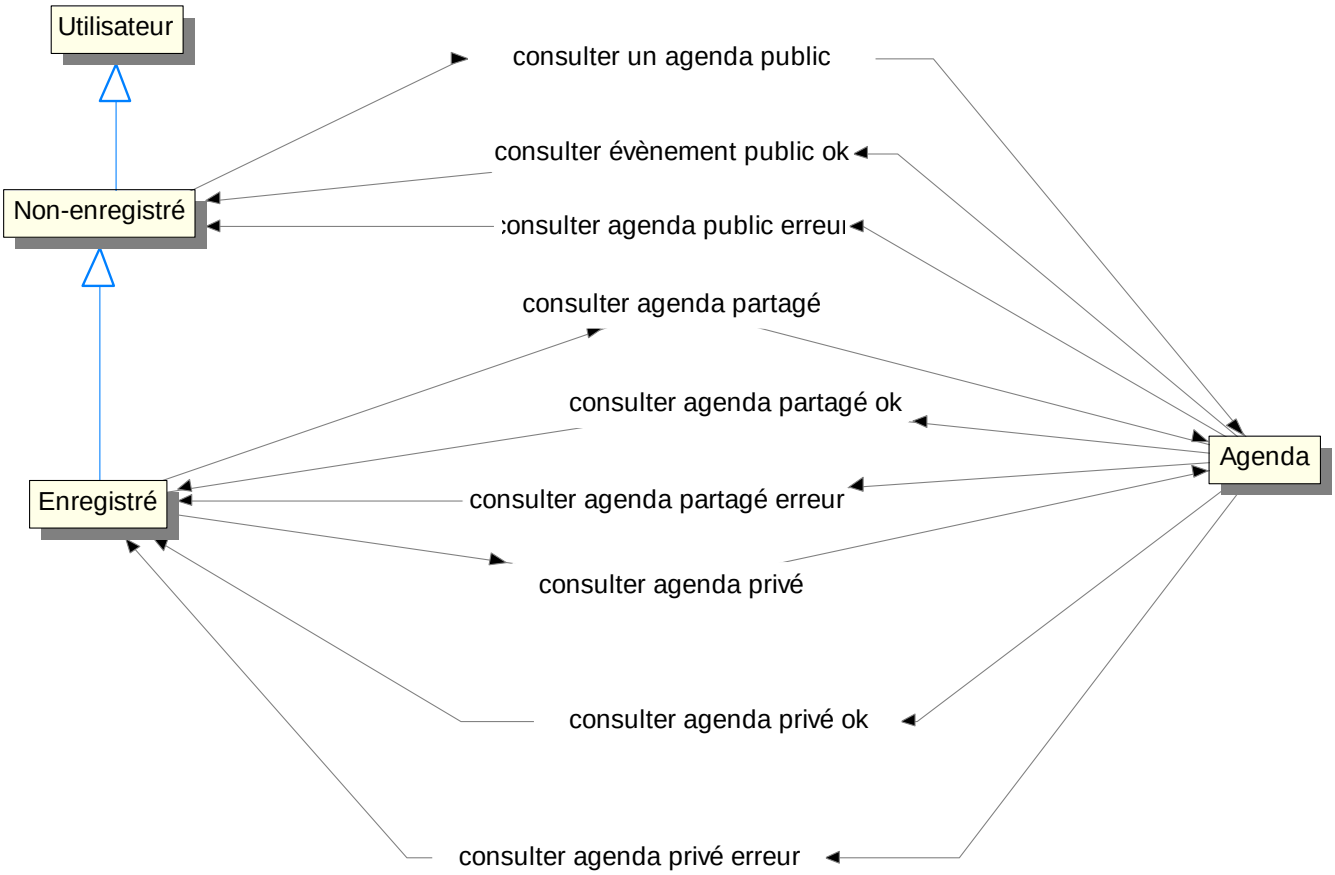
Nous avons choisi pour les diagrammes de flux de différencier les utilisateurs enregistrés des non-enregistrés. Étant donné que ces premiers bénéficient au minimum des possibilités de ces seconds, nous avons choisi d'utiliser une spécialisation. *Agenda* représente le système d'informations dans son ensemble.

Pour chacun des diagrammes, nous considérons que les opérations *créer*, *modifier* et *supprimer*, sont contenues dans l'opération *gérer*.

Ce premier diagramme permet de visualiser les flux qui concernent les différentes manipulations qui ont lieu directement sur les agendas :



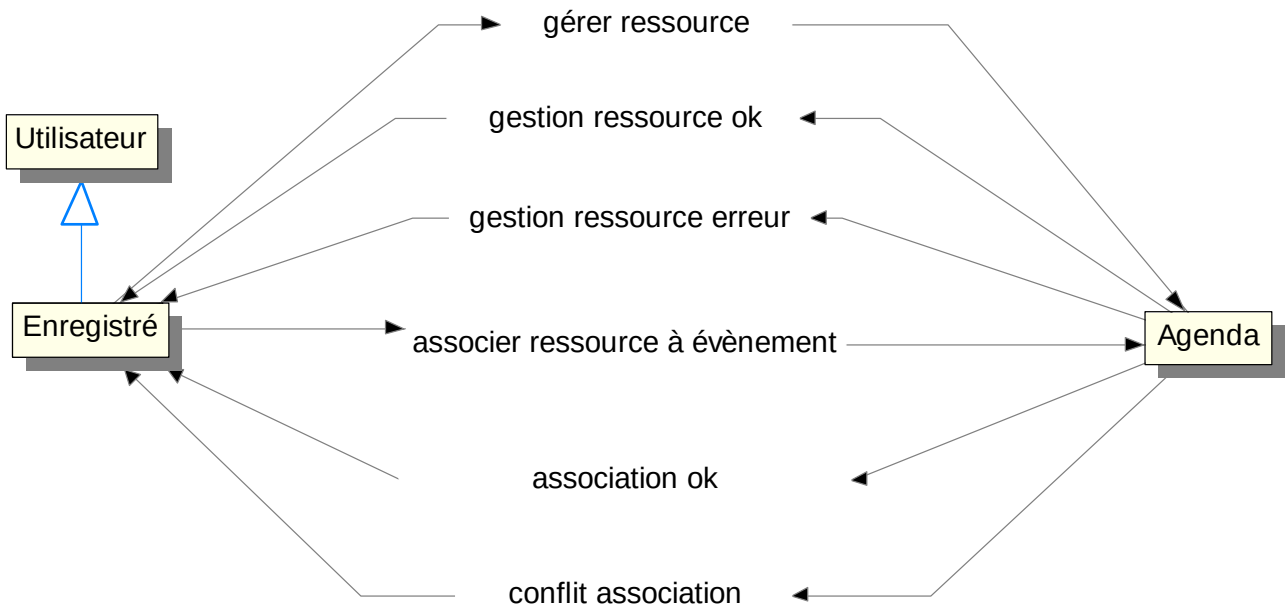
Ce second détaille les différentes possibilités de consultation des agendas, selon le type d'utilisateur et la visibilité de l'agenda :



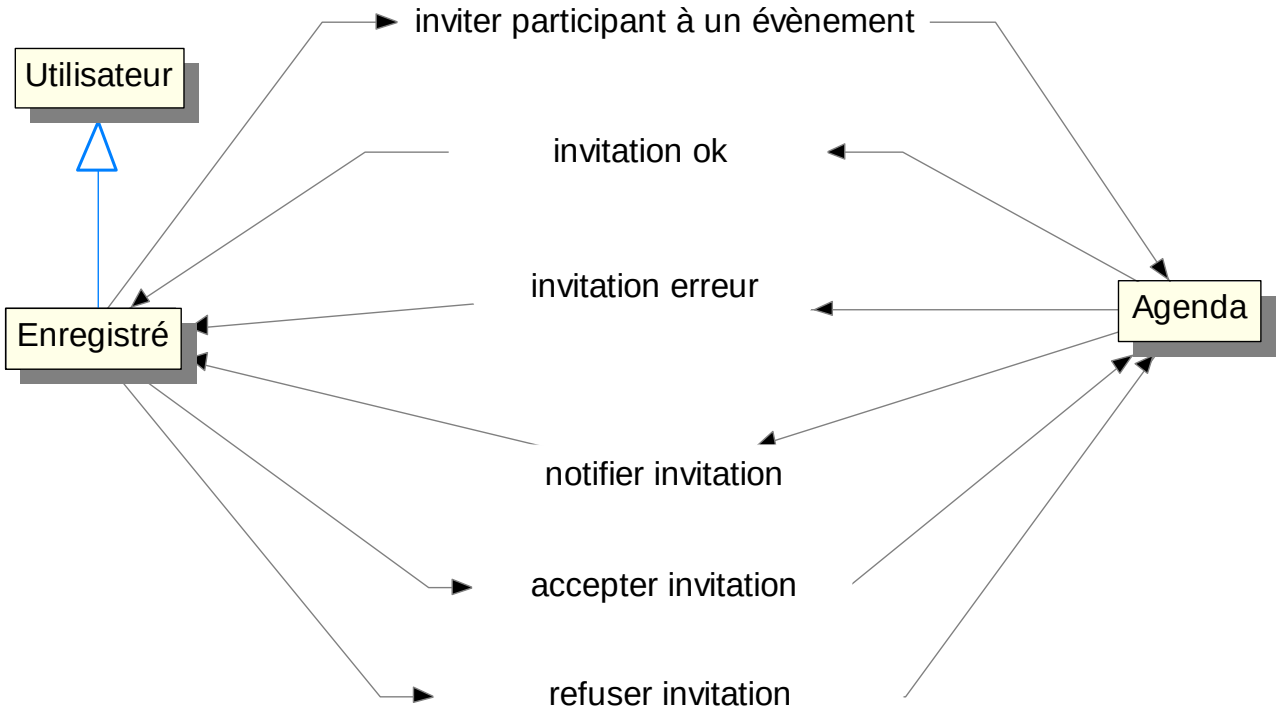
Un acteur *Cron* a ensuite été ajouté pour représenter la machine qui se chargera de déclencher l'envoi des notifications pour les rappels programmés des agendas. À noter qu'une commande UNIX *at* serait plus appropriée, étant donné le caractère ponctuel du rappel.

Ce troisième diagramme précise les flux qui concernent la consultation des événements, selon la visibilité de celui-ci en fonction du type d'utilisateur. Un événement partagé est un événement ajouté par un utilisateur pour lequel l'agenda a été partagé.

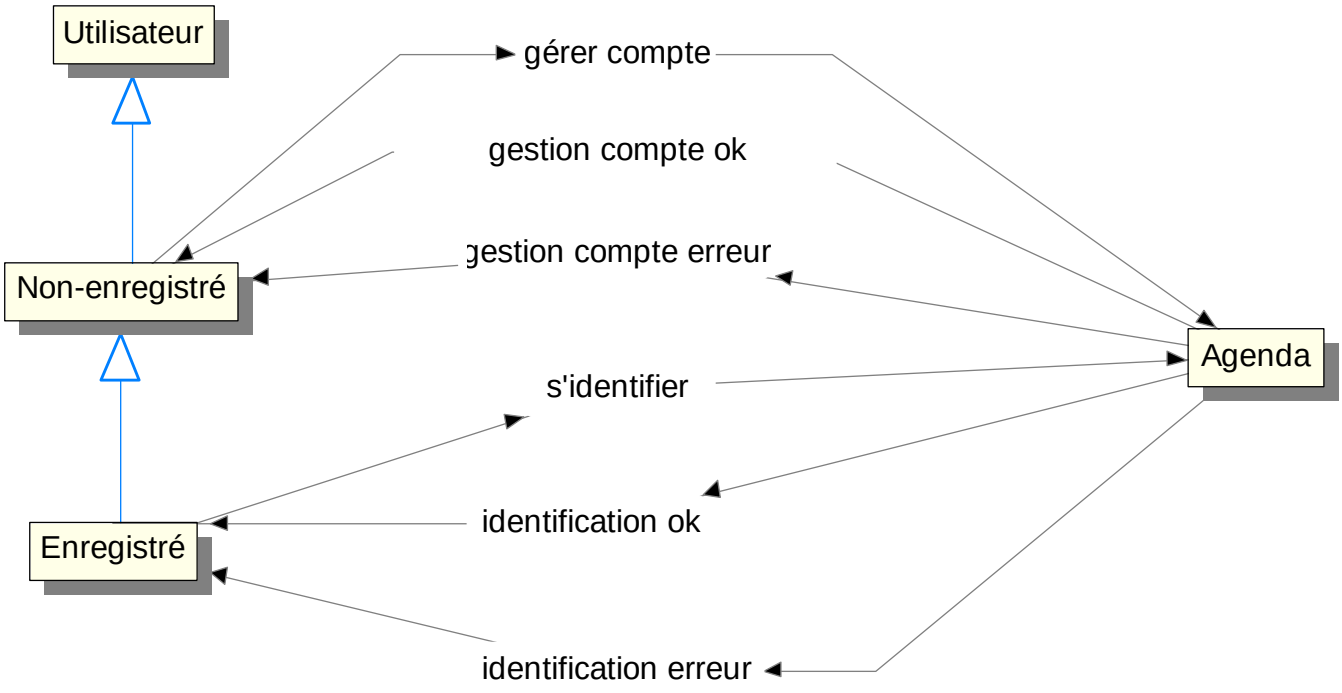
Cet autre diagramme offre la possibilité de visualiser les flux qui concernent la gestion des ressources :



Ce diagramme de flux se concentre sur les flux liés aux échanges qui ont lieu lorsqu'une invitation à un événement est demandée :



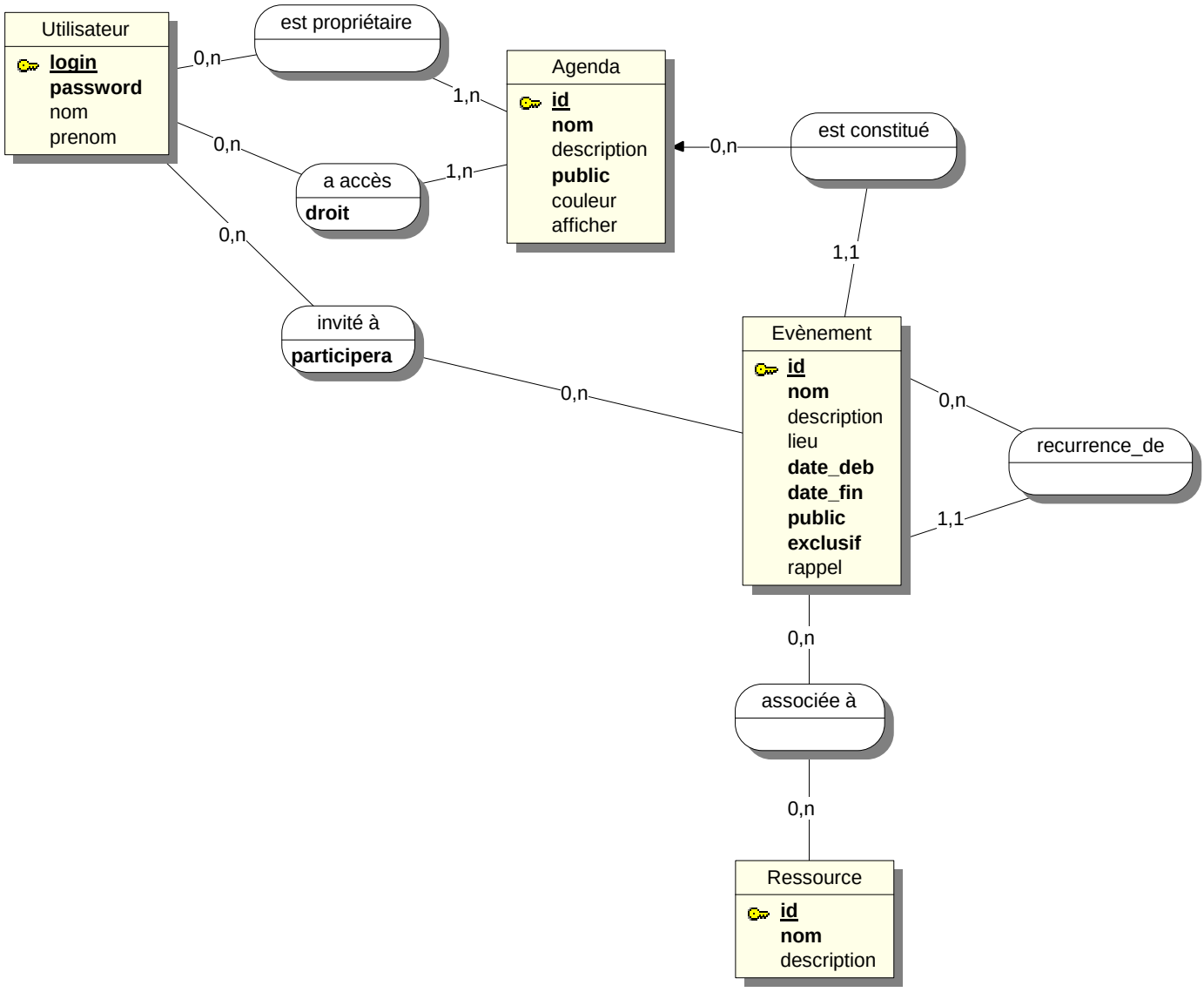
Enfin, ce dernier diagramme permet de visualiser les flux de gestion des comptes utilisateurs et de l'authentification :



1.2 Modèle conceptuel des données

1.2.1 Schéma

Proposition de conception pour l'organisation des données :



1.2.2 Détails

Entités

Nous avons choisi de restreindre notre modèle à quatre entités, qui correspondent aux éléments principaux à prendre en compte dans notre conception : *Utilisateur*, *Agenda*, *Evènement* et *Ressource*.

L'entité *Utilisateur* contient le minimum nécessaire pour identifier un utilisateur. Seuls les attributs *login* (son adresse de courriel) et *password* sont obligatoires. Le *password* est stocké sous forme de *hash md5*.

L'entité *Agenda* contient ce qui est nécessaire pour stocker les paramètres d'un agenda. L'attribut couleur contient une couleur sous forme hexadécimale (six caractères), sans le dièse initial. L'attribut *public* est un booléen, positionné à vrai s'il est public (par défaut, un agenda sera privé). Enfin, l'attribut *afficher* indique si les événements de cet agenda seront affichés (booléen positionné à vrai) sur l'agenda général en page d'accueil.

L'entité *Evènement* stocke les informations liées aux différents événements des différents agendas. Les dates *date_deb* et *date_fin* permettent de définir la durée de l'événement. Elles seront recalculées si l'événement est récurrent (cf. relation *recurrence_de* plus bas). Un événement exclusif verra son attribut correspondant positionné à vrai. Enfin, l'attribut *rappel* contiendra un *timestamp* UNIX pour indiquer le nombre de secondes avant *date_deb* qu'il faudra considérer pour faire le rappel (e.g. si un *timestamp* indique le jour *02/01/1970 00 :00 :00*, il faudra lancer un rappel deux jours avant).

Enfin, l'entité *Ressource* permet de stocker de façon simple les différentes ressources existantes.

Relations

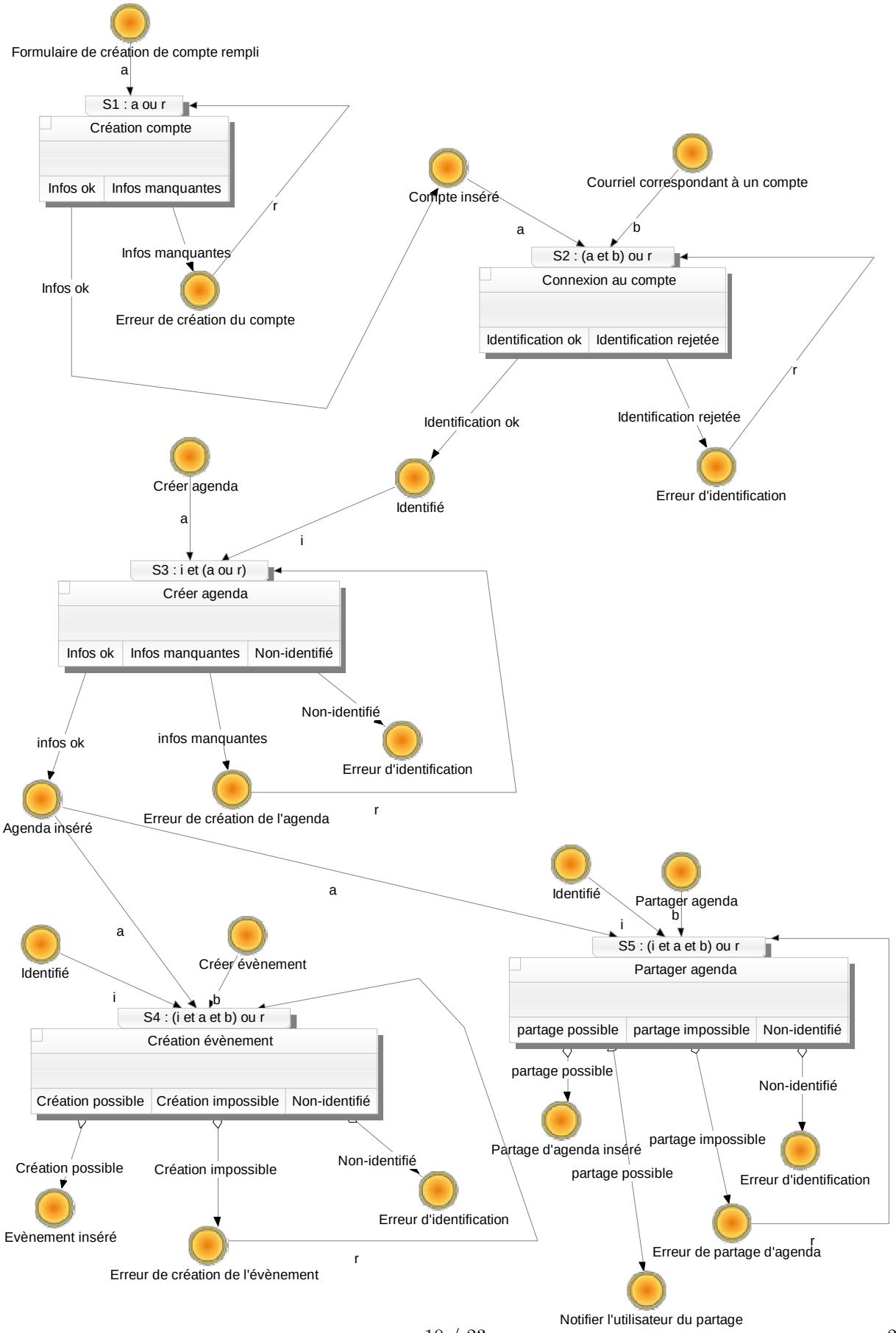
Notre première relation *est_proprietaire* permet de lister facilement les agendas que l'utilisateur a créé. La relation suivant *a_accès*, par l'intermédiaire de son attribut *droit*, permet de préciser ce que l'utilisateur peut faire sur l'agenda. Ces droits utilisent la même notation que les droits UNIX, à l'exception près que le drapeau destiné à l'exécution est remplacé par le partage. Ainsi, en considérant *{4 = consulter, 2 = modifier, 1 = partager}*, un droit de 6 permettra de consulter et modifier, mais pas de partager. Dans le cas où l'utilisateur est propriétaire, le droit ne sera pas consulté et sera considéré comme étant à 7. De la même façon, un droit de 2n'aura pas de sens (on peut difficilement modifier sans consulter).

La participation ou non d'un utilisateur à un événement sera conditionnée par l'attribut *participera* de la relation *invité_à*. Elle sera complétée chaque fois qu'un utilisateur acceptera une invitation.

La composition de l'agenda sera précisée par la relation *est_constitué*, qui indiquera la liste des événements associés. Un événement pouvant être associé à une ou plusieurs ressources, la relation *associé_à* permet d'en connaître la liste.

Enfin, la relation *recurrence_de* qui boucle sur l'entité *Evènement* permet de connaître l'événement source en cas de récurrence. Par exemple, si un événement est créé avec un *date_deb* un samedi et un *rappel* qui contient *W* (toutes les semaines), tous les événements correspondants seront ajoutés à la base de données jusqu'à la date indiquée lors de la saisie. Si l'utilisateur décide finalement de supprimer cette récurrence, ce sera l'attribut *id_recurrence* généré par cette relation qui indiquera ceux qui sont liés, et qui seront donc à supprimer en cascade. Nous avons hésité entre cette approche (créer tous les événements de la récurrence dans la base) ou ne créer que le premier et préciser l'information de récurrence dans ses attributs. Nous avons décidé que cette seconde solution serait trop lourde à gérer pour le serveur (pour chaque affichage de l'agenda, il faudrait donc rechercher en arrière tous les événements qui ont une récurrence pour vérifier s'il faut les afficher).

1.3 Modèle conceptuel des traitements

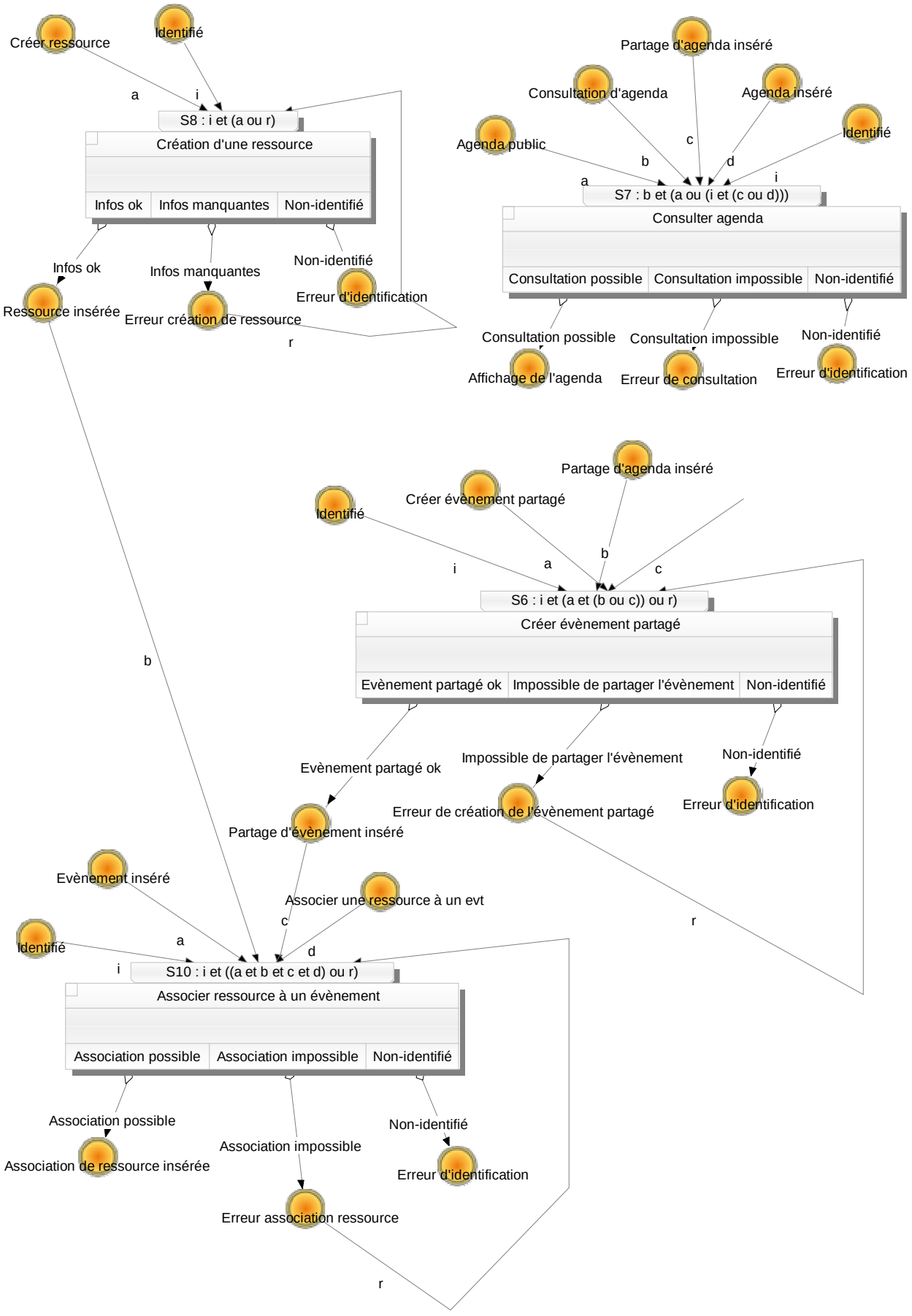


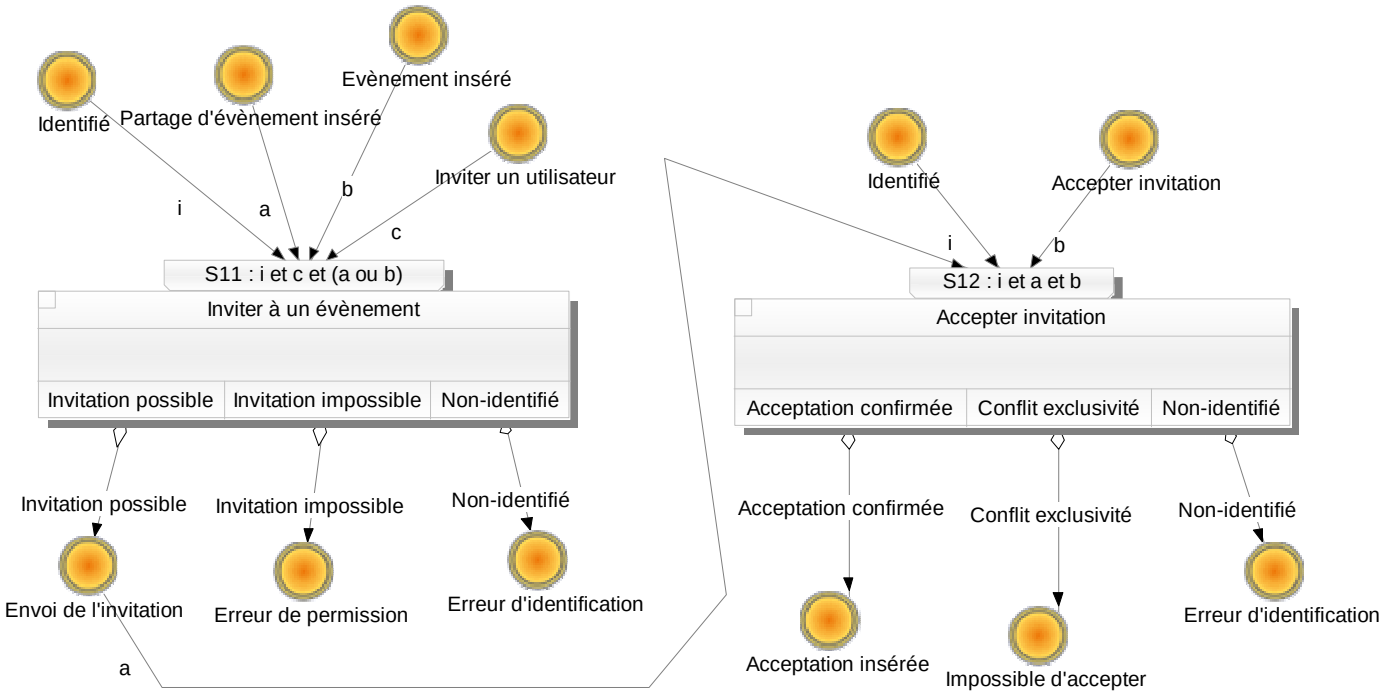
Afin de préciser les conditions nécessaires pour effectuer les opérations les plus importantes, nous avons réalisé quelques diagrammes MCT.

Lorsque la lisibilité était gênée, nous avons préféré créer des duplicatas des événements plutôt que matérialiser les enchainements.

Afin de ne pas surcharger le diagramme, nous avons omis les messages. La plupart des opérations débouchent sur une insertion, qui implique toujours l'affichage d'une page de confirmation.

Lorsque des erreurs ont lieu (conditions r), l'utilisateur est presque toujours redirigé vers la page d'où provient l'erreur, qui se chargera alors de préciser le pourquoi de ce retour. Dans le cas d'une erreur d'identification (conditions i), l'utilisateur est redirigé vers la page de connexion (cf. l'événement *Erreur d'identification* originel, de l'opération *Connexion au compte*).


Créer ressource
Identifié
a
i
S8 : i et (a ou r)
Création d'une ressource
Infos ok
Infos manquantes
Non-identifié
Infos ok
Infos manquantes
Non-identifié
Ressource insérée
Erreur création de ressource
Erreur d'identification
r
Partage d'agenda inséré
Consultation d'agenda
Agenda inséré
Identifié
Agenda public
b
c
d
a
i
S7 : b et (a ou (i et (c ou d)))
Consulter agenda
Consultation possible
Consultation impossible
Non-identifié
Consultation possible
Consultation impossible
Non-identifié
Affichage de l'agenda
Erreur de consultation
Erreur d'identification
Partage d'agenda inséré
Créer évènement partagé
Identifié
i
a
b
c
S6 : i et (a et (b ou c)) ou r
Créer évènement partagé
Evènement partagé ok
Impossible de partager l'évènement
Non-identifié
b
Evènement partagé ok
Impossible de partager l'évènement
Non-identifié
Partage d'évènement inséré
Erreur de création de l'évènement partagé
Erreur d'identification
r
Evènement inséré
Associer une ressource à un evt
c
Identifié
a
d
i
S10 : i et ((a et b et c et d) ou r)
Associer ressource à un évènement
Association possible
Association impossible
Non-identifié
Association possible
Association impossible
Non-identifié
Association de ressource insérée
Erreur association ressource
Erreur d'identification
r



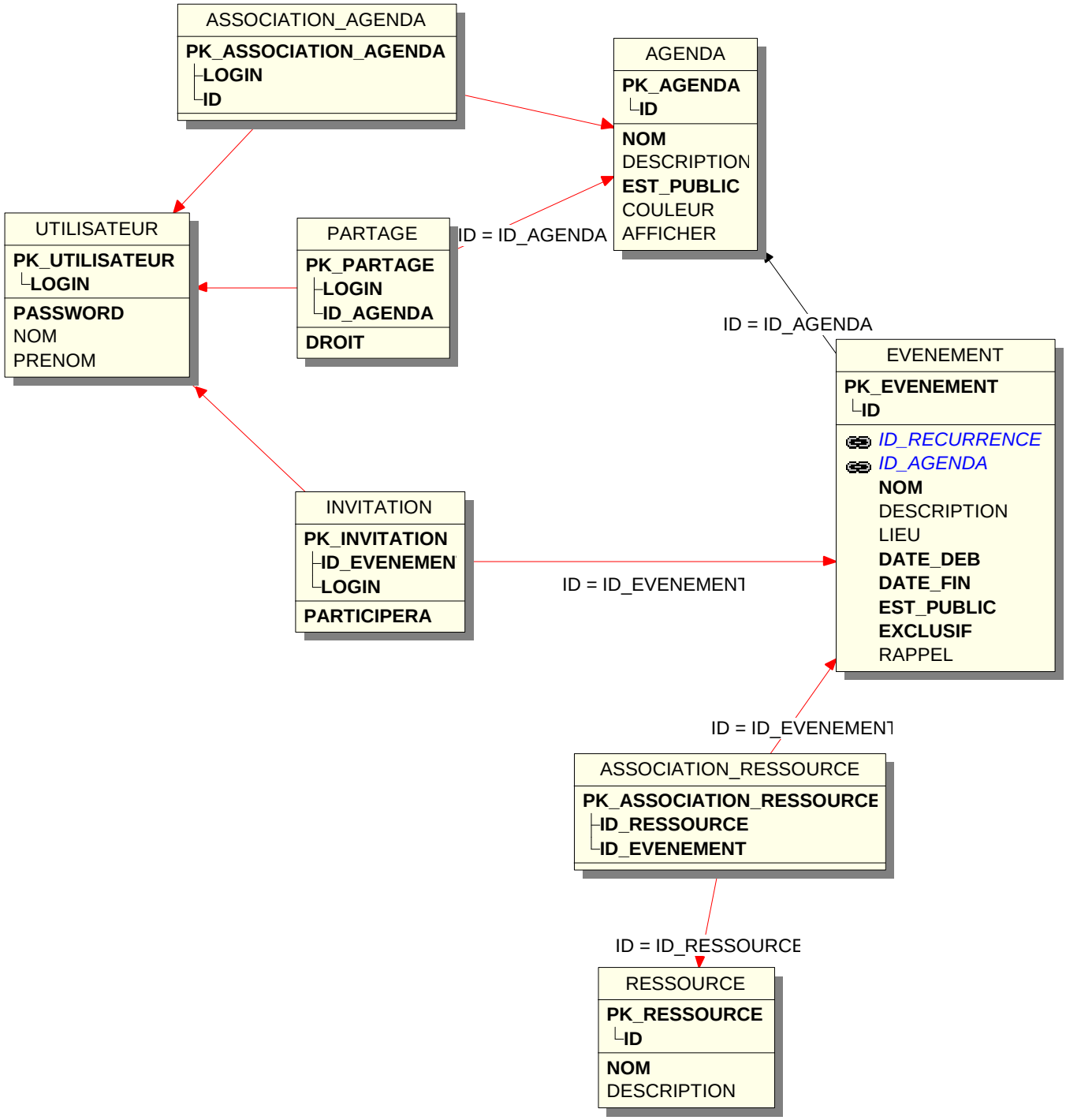
Lorsqu'un événement d'entrée correspond à *X créé*, il s'agit du duplicata d'un événement de sortie d'une autre opération. La condition correspondante signifie alors que l'utilisateur est propriétaire du X (il l'a créé).

Les erreurs telles que *Erreur de consultation* ne donnent pas lieu à un renvoi vers l'opération précédente, puisqu'il y a un risque de boucle. Nous considérons donc qu'un simple message d'erreur s'affiche.

2 Modèles logiques

2.1 Modèle logique relationnel

Proposition de base de données, déduite du MCD :



2.2 Modèle logique des traitements

2.2.1 Description lexicale

Cette description lexicale a pour but de préciser les différentes opérations du MCT. Pour chacune d'entre elles, nous avons proposé une description, et nous avons détaillé les entrées, sorties et sous-opérations qui aideront à son implémentation.

L'identification de l'utilisateur est testée à chaque page, et n'apparaît pas dans les sous-opérations pour plus de clarté.

Id	**Nom opération**	**Description**	**Entrées**	**Sorties**	**Sous opérations**
S1	Création compte	Créer un compte utilisateur dans le système	– Login utilisateur – Mot de passe – Nom – Prénom	– Code de succès ou d'erreur	– Afficher le formulaire d'inscription – Vérifier les champs du formulaire – Informer l'utilisateur des erreurs – Vérifier si le login n'existe pas dans la base – Insérer le nouvel utilisateur dans la base – Afficher le succès et envoyer un mail de confirmation
S2	Connexion au compte	Connecter un utilisateur au système	– Login utilisateur – Mot de passe	– Code de succès ou d'erreur – Session	– Afficher le formulaire de connexion – Vérifier les champs de formulaire – Vérifier que le mot de passe et le login sont dans la base – Afficher les erreurs de connexion et recharger le formulaire – Connecter l'utilisateur et créer une session
S3	Créer agenda	Ajouter un nouvel agenda dans le système	– Nom d'agenda – Description – Lieu – Public ou privé – Couleur – Afficher dans les menus	– Id agenda	– Afficher le formulaire de création – Vérifier les champs du formulaire – Afficher les erreurs et recharger le formulaire – Insérer l'agenda dans la base

S4	Création événement	Ajouter un nouvel événement dans le système	– Id agenda – Nom événement – Description – Lieu – Date début et date fin – Public ou privé – Exclusif ou non – Période de rappel	– Id événement	– Afficher le formulaire de création – Vérifier les champs du formulaire – Afficher les erreurs et recharger le formulaire – Insérer l'événement dans la base
S5	Partager agenda	Partager un agenda avec un utilisateur du système	– Id agenda – Id utilisateur – Droits accordés		– Afficher le formulaire de partage (utilisateurs, droits) – Vérifier les champs du formulaire – Afficher les erreurs et recharger le formulaire – Insérer le partage dans le système et envoyer un courriel aux utilisateurs
S6	Créer événement partagé	Ajouter un événement dans un agenda partagé	– Id agenda – Nom événement – Description – Lieu – Date début et date fin – Public ou privé – Exclusif ou non – Période de rappel	– Id événement	– Afficher le formulaire de création d'un événement – Vérifier les champs du formulaire – Insérer l'événement dans tous les agendas
S7	Consulter agenda	Consulter la liste des événements d'un agenda	– Id agenda	– Liste d'événements	– Vérifier les droits de l'utilisateur pour l'agenda – Récupérer l'agenda dans la base – Afficher erreur si problème de droit ou agenda inexistant – Afficher tous les événements de l'agenda pour une période donnée

S8	Création d'une ressource	Créer une ressource dans le système	– Description ressource	– Id ressource	– Afficher le formulaire d'ajout d'une ressource – Vérifier les champs du formulaire – Afficher erreur et recharger le formulaire – Insérer la ressource dans la base
S10	Associer ressource à un événement	Ajouter une association entre un événement et une ressource dans le système	– Id ressource – Id événements		– Afficher la liste des ressources – Vérifier si la ressource n'est pas associée à un autre événement – Insérer l'association dans la base et réserver la ressource
S11	Inviter à un événement	Inviter un autre utilisateur à participer à un événement	– Id événement – Id utilisateur		– Afficher le formulaire d'invitation – Insérer l'invitation dans la base et envoyer un mail aux invités
S12	Accepter invitation	Accepter l'invitation d'un utilisateur	– Id événement – Id utilisateur – Participera ou non	– Participera ou non	– Afficher la page de présentation d'un événement – Insérer la réponse de l'utilisateur dans la base

2.2.2 Algèbre relationnelle

Afin d'aller plus loin dans le lien entre la conception et l'implémentation, nous avons utilisé de l'algèbre relationnelle pour quelques opérations, en particulier pour définir l'affichage de l'accueil (liste des événements tirés des différents agendas, avec les couleurs associées, sur le mois en cours).

La notion de recherche dans les événements correspond à l'opération *getEvenements*, avec les différents filtres supplémentaires dans la projection.

getEvenements

INPUT

`id_agenda` agenda depuis lequel récupérer les événements

`date_deb` date à partir de laquelle rechercher

`date_fin` date jusqu'à laquelle rechercher

`prive` booléen indiquant si les événements privés doivent être retournés

OUTPUT

`evts` liste de *Evenement <id,id_agenda,nom,description,lieu,date_deb,date_fin,est_public,exclusif,rappel>*

Begin

evts = $\sigma_{id=id\_agenda}$(**Evenement**)
evts = $\pi_{id,id\_agenda,nom,description,lieu,date\_deb,date\_fin,est\_public,exclusif,rappel}(evts)$

```
if(!prive) {
    evts = σ_{public=true}(evts)
}

return evts
```

End

getAgendas

INPUT

`id_user` login de l'utilisateur

`cachés` booléen indiquant si les agendas cachés doivent être retournés

OUTPUT

`agendas` liste de *Agenda <id,nom,description,est_public,couleur,afficher>*

Begin

agendas = $\pi_{id\_agenda}(\sigma_{login=id\_user}$(**Association_Agenda**))

```
foreach(a in agendas) {
    liste.add(π_{id,nom,description,est_public,couleur,afficher}(σ_{id=a.id}(Agenda)))
}

return liste
```

End

afficheAccueil

INPUT

`id_user` login de l'utilisateur

`cachés` booléen indiquant si les agendas cachés doivent être retournés

OUTPUT

Aucun.

Begin

```
agendas = getAgendas(id_user, false)

foreach(a in agendas) {
    evts = getEvenements(a.id, today(), endOfTheMonth()))
```

```
    foreach(e in evts) {
        foreach(j in monthDays()) {
            if(j > e.date_deb && j < e.date_fin) {
                evtsByDay[j].push([ e, a ])
            }
        }
    }
}

foreach(j in monthDays()) {
    if(evtsByDay[j] != null) {
        foreach(ea in evtsByDay[j]) {
            areaColor(ea.a.couleur)
            displayArea(j : ea.e.nom)
        }
    } else {
        areaColor(black)
        displayArea(j)
    }
}
```

End

afficherEvt

INPUT

`id_evt` événement à afficher

OUTPUT

Aucun.

Begin

`evt` = $\pi_{id,id\_agenda,nom,description,lieu,date\_deb,date\_fin,est\_public,exclusif,rappel}(\sigma_{id=id\_evt}(\texttt{Evenement}))$

```
display(evt)
```

End

identification

INPUT

`id_user` login de l'utilisateur

`password` mot de passe de l'utilisateur

OUTPUT

`identifié` booléen indiquant si l'identification a réussie ou non

Begin

```
passHash = md5(password)
utilisateur = σ_{login=id_user,password=passHash}(Utilisateur)
nb_utilisateur = ⋈(utilisateur)

if(identifié = (nb_utilisateur > 0)) {
    utilisateur = π_{login,nom,prenom}(utilisateur)

    create_session([utilisateur.login, utilisateur.nom, utilisateur.prenom])
}

return identifié

End
```

3 SQL

3.1 Tables

Tables SQL déduites du MLR :

```
1  DROP TABLE UTILISATEUR CASCADE CONSTRAINTS;
2
3  DROP TABLE EVENEMENT CASCADE CONSTRAINTS;
4
5  DROP TABLE AGENDA CASCADE CONSTRAINTS;
6
7  DROP TABLE RESSOURCE CASCADE CONSTRAINTS;
8
9  DROP TABLE PARTAGE CASCADE CONSTRAINTS;
10
11 DROP TABLE ASSOCIATION_RESSOURCE CASCADE CONSTRAINTS;
12
13 DROP TABLE ASSOCIATION_AGENDA CASCADE CONSTRAINTS;
14
15 DROP TABLE INVITATION CASCADE CONSTRAINTS;
16
17 -- ------------------------------------------------------------------------
18 --         CREATION DE LA BASE
19 -- ------------------------------------------------------------------------
20
21 CREATE DATABASE ajuda_MLR;
22
23 -- ------------------------------------------------------------------------
24 --         TABLE : UTILISATEUR
25 -- ------------------------------------------------------------------------
26
27 CREATE TABLE UTILISATEUR
28     (
29      LOGIN CHAR(32)   NOT NULL,
30      PASSWORD CHAR(32)   NOT NULL,
31      NOM CHAR(32)   NULL,
32      PRENOM CHAR(32)   NULL
33  ,   CONSTRAINT PK_UTILISATEUR PRIMARY KEY (LOGIN)
34     ) ;
35
36 -- ------------------------------------------------------------------------
37 --         TABLE : EVENEMENT
38 -- ------------------------------------------------------------------------
39
40 CREATE TABLE EVENEMENT
41     (
42      ID NUMBER(5)   NOT NULL,
43      ID_AGENDA NUMBER(5)   NOT NULL,
44      ID_RECURRENCE NUMBER(5)   NOT NULL,
45      NOM CHAR(32)   NOT NULL,
46      DESCRIPTION CLOB   NULL,
47      LIEU CLOB   NULL,
48      DATE_DEB DATE   NOT NULL,
49      DATE_FIN DATE   NOT NULL,
```

```
50        EST_PUBLIC NUMBER(1)   NOT NULL,
51        EXCLUSIF NUMBER(1)   NOT NULL,
52        RAPPEL DATE   NULL
53  ,     CONSTRAINT PK_EVENEMENT PRIMARY KEY (ID)
54      ) ;
55
56  -- ------------------------------------------------------------------------
57  --          INDEX DE LA TABLE EVENEMENT
58  -- ------------------------------------------------------------------------
59
60  CREATE   INDEX I_FK_EVENEMENT_AGENDA
61         ON EVENEMENT (ID_AGENDA ASC)
62        ;
63
64  -- ------------------------------------------------------------------------
65  --          TABLE : AGENDA
66  -- ------------------------------------------------------------------------
67
68  CREATE TABLE AGENDA
69      (
70        ID NUMBER(5)   NOT NULL,
71        NOM CHAR(32)   NOT NULL,
72        DESCRIPTION CLOB   NULL,
73        EST_PUBLIC NUMBER(1)   NOT NULL,
74        COULEUR CHAR(6)   NULL,
75        AFFICHER NUMBER(1)   NULL
76  ,     CONSTRAINT PK_AGENDA PRIMARY KEY (ID)
77      ) ;
78
79  -- ------------------------------------------------------------------------
80  --          TABLE : RESSOURCE
81  -- ------------------------------------------------------------------------
82
83  CREATE TABLE RESSOURCE
84      (
85        ID NUMBER(5)   NOT NULL,
86        NOM CHAR(32)   NOT NULL,
87        DESCRIPTION CLOB   NULL
88  ,     CONSTRAINT PK_RESSOURCE PRIMARY KEY (ID)
89      ) ;
90
91  -- ------------------------------------------------------------------------
92  --          TABLE : PARTAGE
93  -- ------------------------------------------------------------------------
94
95  CREATE TABLE PARTAGE
96      (
97        LOGIN CHAR(32)   NOT NULL,
98        ID NUMBER(5)   NOT NULL,
99        DROIT NUMBER(2)   NOT NULL
100 ,     CONSTRAINT PK_PARTAGE PRIMARY KEY (LOGIN, ID)
101     ) ;
102
103 -- ------------------------------------------------------------------------
104 --          INDEX DE LA TABLE PARTAGE
105 -- ------------------------------------------------------------------------
106
107 CREATE   INDEX I_FK_PARTAGE_UTILISATEUR
108        ON PARTAGE (LOGIN ASC)
109       ;
110
111 CREATE   INDEX I_FK_PARTAGE_AGENDA
112        ON PARTAGE (ID ASC)
113       ;
114
115 -- ------------------------------------------------------------------------
116 --          TABLE : ASSOCIATION_RESSOURCE
117 -- ------------------------------------------------------------------------
118
119 CREATE TABLE ASSOCIATION_RESSOURCE
120     (
121       ID_EVENEMENT NUMBER(5)   NOT NULL,
122       ID_RESSOURCE NUMBER(5)   NOT NULL
123 ,     CONSTRAINT PK_ASSOCIATION_RESSOURCE PRIMARY KEY (ID_EVENEMENT, ID_RESSOURCE)
124     ) ;
125
126 -- ------------------------------------------------------------------------
127 --          INDEX DE LA TABLE ASSOCIATION_RESSOURCE
128 -- ------------------------------------------------------------------------
129
130 CREATE   INDEX I_FK_ASSOCIATION_RESSOURCE_RES
131        ON ASSOCIATION_RESSOURCE (ID_EVENEMENT ASC)
132       ;
133
```

```
134 CREATE  INDEX I_FK_ASSOCIATION_RESSOURCE_EVE
135       ON ASSOCIATION_RESSOURCE (ID_RESSOURCE ASC)
136       ;
137
138 -- ------------------------------------------------------------------------
139 --        TABLE : ASSOCIATION_AGENDA
140 -- ------------------------------------------------------------------------
141
142 CREATE TABLE ASSOCIATION_AGENDA
143     (
144      LOGIN CHAR(32)  NOT NULL,
145      ID_AGENDA NUMBER(5)  NOT NULL
146  ,    CONSTRAINT PK_ASSOCIATION_AGENDA PRIMARY KEY (LOGIN, ID_AGENDA)
147     ) ;
148
149 -- ------------------------------------------------------------------------
150 --        INDEX DE LA TABLE ASSOCIATION_AGENDA
151 -- ------------------------------------------------------------------------
152
153 CREATE  INDEX I_FK_ASSOCIATION_AGENDA_UTILIS
154       ON ASSOCIATION_AGENDA (LOGIN ASC)
155       ;
156
157 CREATE  INDEX I_FK_ASSOCIATION_AGENDA_AGENDA
158       ON ASSOCIATION_AGENDA (ID_AGENDA ASC)
159       ;
160
161 -- ------------------------------------------------------------------------
162 --        TABLE : INVITATION
163 -- ------------------------------------------------------------------------
164
165 CREATE TABLE INVITATION
166     (
167      ID NUMBER(5)  NOT NULL,
168      LOGIN CHAR(32)  NOT NULL,
169      PARTICIPERA NUMBER(1)  NOT NULL
170  ,    CONSTRAINT PK_INVITATION PRIMARY KEY (ID, LOGIN)
171     ) ;
172
173 -- ------------------------------------------------------------------------
174 --        INDEX DE LA TABLE INVITATION
175 -- ------------------------------------------------------------------------
176
177 CREATE  INDEX I_FK_INVITATION_EVENEMENT
178       ON INVITATION (ID ASC)
179       ;
180
181 CREATE  INDEX I_FK_INVITATION_UTILISATEUR
182       ON INVITATION (LOGIN ASC)
183       ;
184
185
186 -- ------------------------------------------------------------------------
187 --        CREATION DES REFERENCES DE TABLE
188 -- ------------------------------------------------------------------------
189
190
191 ALTER TABLE EVENEMENT ADD (
192       CONSTRAINT FK_EVENEMENT_AGENDA
193             FOREIGN KEY (ID_AGENDA)
194                   REFERENCES AGENDA (ID))    ;
195
196 ALTER TABLE EVENEMENT ADD (
197     CONSTRAINT FK_EVENEMENT_RECURRENCE
198             FOREIGN KEY (ID_RECURRENCE)
199                   REFERENCES EVENEMENT (ID))    ;
200
201 ALTER TABLE PARTAGE ADD (
202       CONSTRAINT FK_PARTAGE_UTILISATEUR
203             FOREIGN KEY (LOGIN)
204                   REFERENCES UTILISATEUR (LOGIN))    ;
205
206 ALTER TABLE PARTAGE ADD (
207       CONSTRAINT FK_PARTAGE_AGENDA
208             FOREIGN KEY (ID)
209                   REFERENCES AGENDA (ID))    ;
210
211 ALTER TABLE ASSOCIATION_RESSOURCE ADD (
212       CONSTRAINT FK_ASSOCIATION_RESSOURCE_RESSO
213             FOREIGN KEY (ID_EVENEMENT)
214                   REFERENCES RESSOURCE (ID))    ;
215
216 ALTER TABLE ASSOCIATION_RESSOURCE ADD (
217       CONSTRAINT FK_ASSOCIATION_RESSOURCE_EVENE
```

```
218             FOREIGN KEY (ID_RESSOURCE)
219                     REFERENCES EVENEMENT (ID))     ;
220
221 ALTER TABLE ASSOCIATION_AGENDA ADD (
222       CONSTRAINT FK_ASSOCIATION_AGENDA_UTILISAT
223             FOREIGN KEY (LOGIN)
224                     REFERENCES UTILISATEUR (LOGIN))     ;
225
226 ALTER TABLE ASSOCIATION_AGENDA ADD (
227       CONSTRAINT FK_ASSOCIATION_AGENDA_AGENDA
228             FOREIGN KEY (ID_AGENDA)
229                     REFERENCES AGENDA (ID))     ;
230
231 ALTER TABLE INVITATION ADD (
232       CONSTRAINT FK_INVITATION_EVENEMENT
233             FOREIGN KEY (ID)
234                     REFERENCES EVENEMENT (ID))     ;
235
236 ALTER TABLE INVITATION ADD (
237       CONSTRAINT FK_INVITATION_UTILISATEUR
238             FOREIGN KEY (LOGIN)
239                     REFERENCES UTILISATEUR (LOGIN))     ;
240
241
242 -- ----------------------------------------------------------------------------
243 --                      FIN DE GENERATION
244 -- ----------------------------------------------------------------------------
```

3.2 Triggers

Après avoir étudié les triggers proposés par Win'Design et constaté du manque de pertinence de ceux-ci, nous avons décidé de ne pas les intégrer à notre conception. Préférer contrôler les différentes contraintes dans le code de l'application plutôt que de s'appuyer sur le SGBD permet de s'affranchir de la nécessité d'utiliser celui-ci. En outre, cette démarche est plus cohérente avec l'utilisation de librairies telles que PDO. Cette base de données n'est pas destinée à être modifiée par d'autres applications que la nôtre.