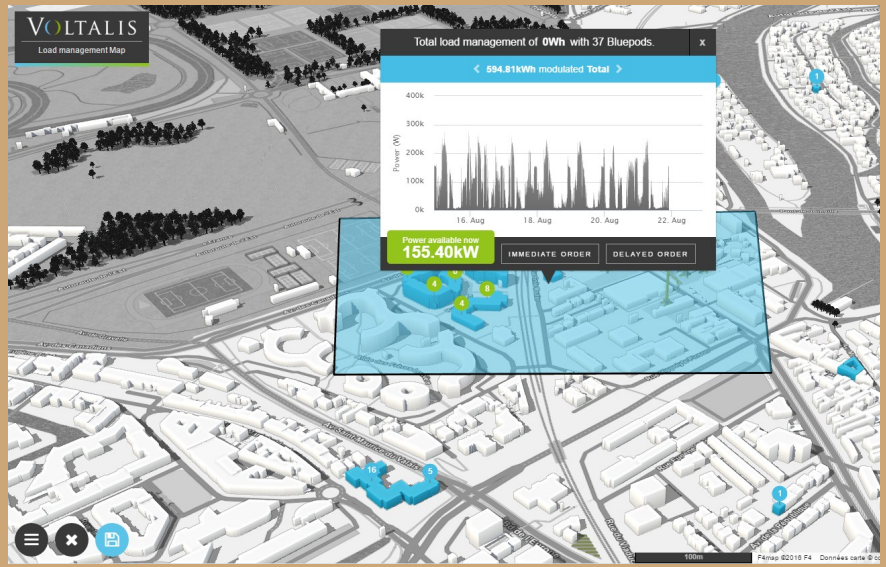
L’application Blue Map

21 octobre 2016

- F4-Group
  - API F4Map/Carte 3D
- Voltalis
  - Effacement diffus
- Blue Map
  - Moteur de recherche
  - Outils de support technique temps réel

Qu’est-ce que c’est ?



Les difficultés de mon stage en entreprise

Débutant en informatique

Technologies non vues en cours

Période estivale, vacances

Le cahier des charges

L’utilisateur peut piloter la carte grâce à un **moteur de recherche** de localités.

Un **support technique temps-réel** utilisable par le **gestionnaire de réseau** afin de pouvoir **communiquer avec l’équipe Voltalis**.

Voir des statistiques de consommation en sélectionnant directement sur la carte

- un Bluepod
- un ensemble de Bluepods en traçant un polygone

Sauvegarder des ensembles de Blue pods

- supprimer les ensembles sauvegardés

La démarche

Inspiré de la démarche UP/UML 2 de Pascal Roques

UML 2 - Modéliser une application web 4ème édition
Aux éditions Eyrolles

“comment passer des besoins utilisateurs au code”.

Réadaptée par rapport au **cours** de **Pascal Buguet** et **à mon projet**.

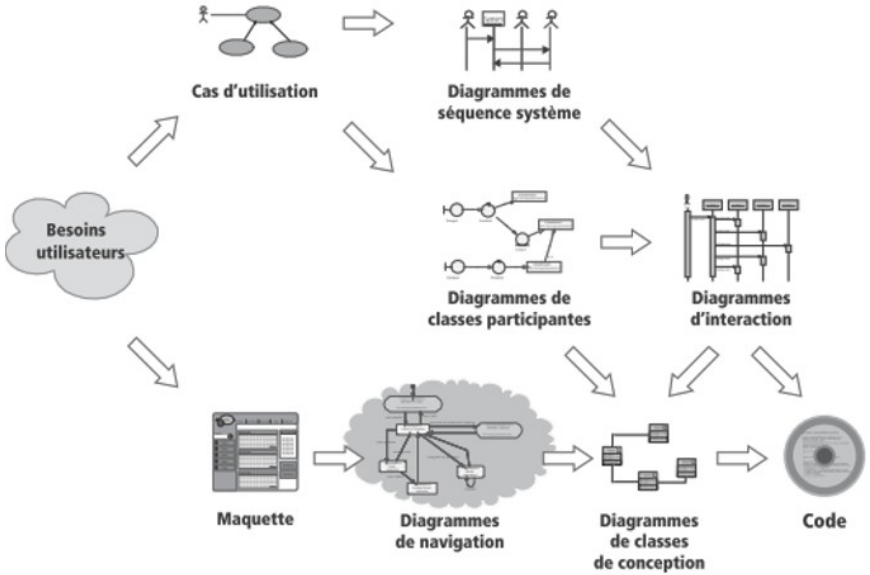


Figure 1–20 Schéma complet du processus de modélisation d'une application web

Les diagrammes de cas d’utilisation

Après l’**expression des besoins**, on les regroupe par **fonctionnalités**.

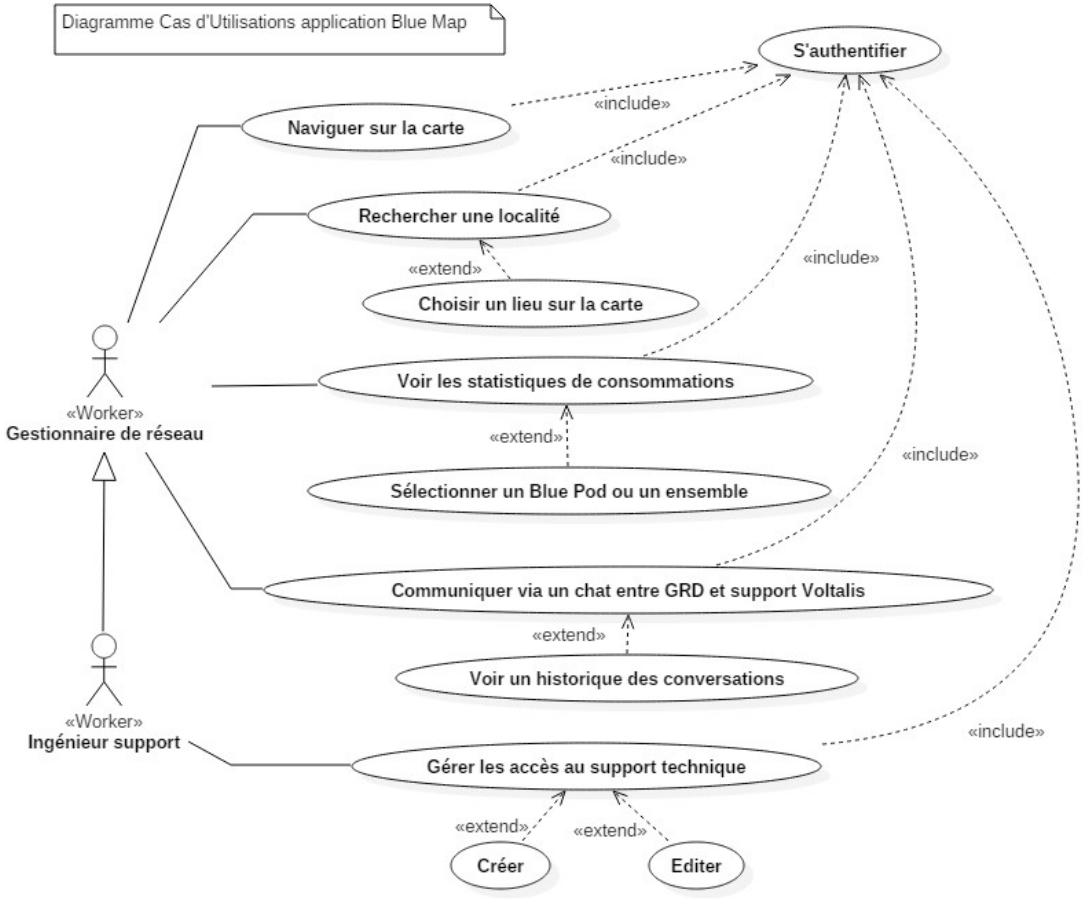
Une **fonctionnalité**, un “use case”

Le **tout** : un DCU

Les cas d’utilisation

Les concepts importants

- CU, DCU
- Acteur
- Héritage
- Fonctionnalité
- Association
- Inclusion, Extension



Les diagrammes d’activité

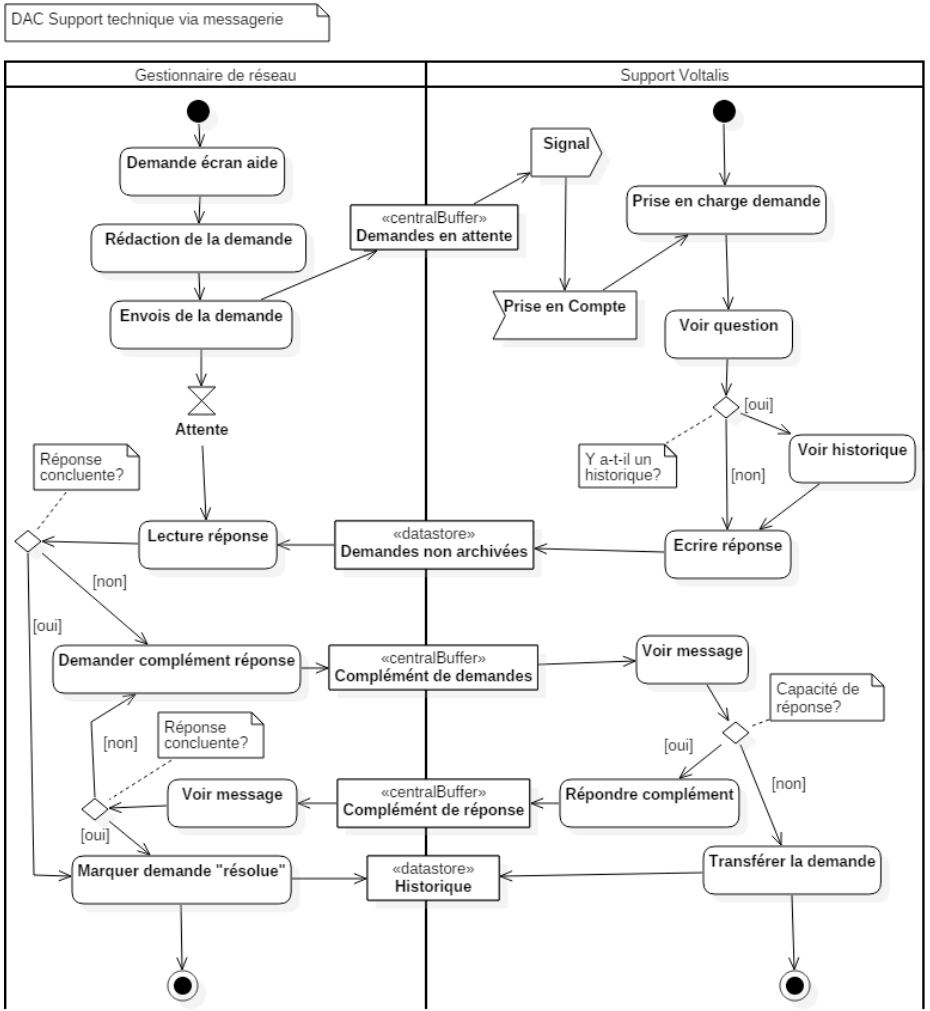
Permet de **compléter** le **cas d’utilisation** communiquer via support technique.

Montre les **différentes action** sur une seule **dimension**, “ à plat ”, et leur déroulement.

Diagramme d’activité

Les concepts importants

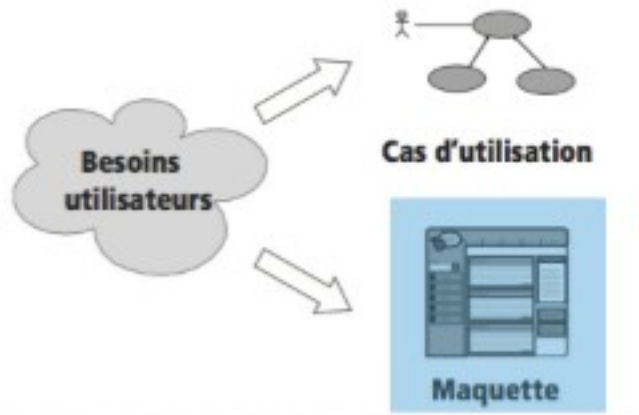
- Acteur
- Point d’entrée
- Action
- Transition
- Flux
- Signal
- Evenement
- Garde



Les maquettes/interfaces

Après l'**expression des besoins**, on prépare des maquettes **à but fonctionnel**, afin de garantir l'utilité et intuitivité.

Les maquettes orientées fonctionnel servent à **se mettre d'accord** entre le **client** et les **développeurs**.



Pourquoi ?

Manque-t-il une fonctionnalité ?

Est-elle
évidente/clair/compréhensible ?

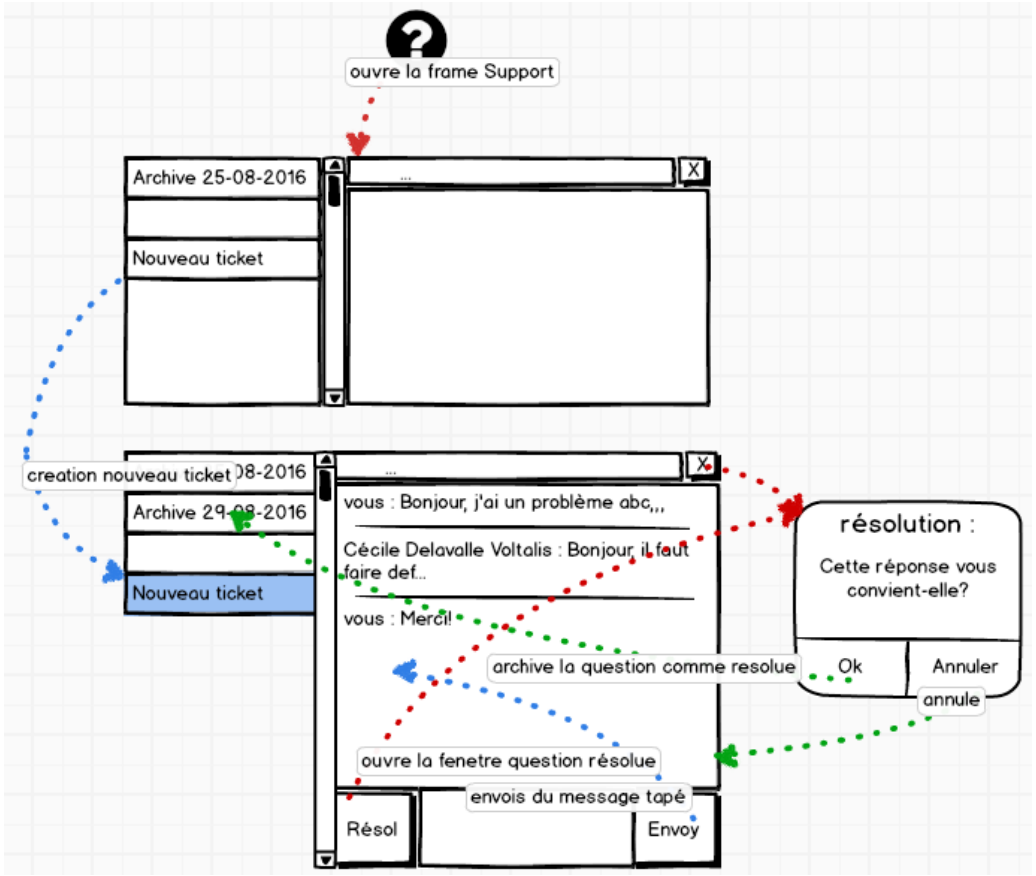
Correspond aux attentes du client ?

Les interfaces et maquettes

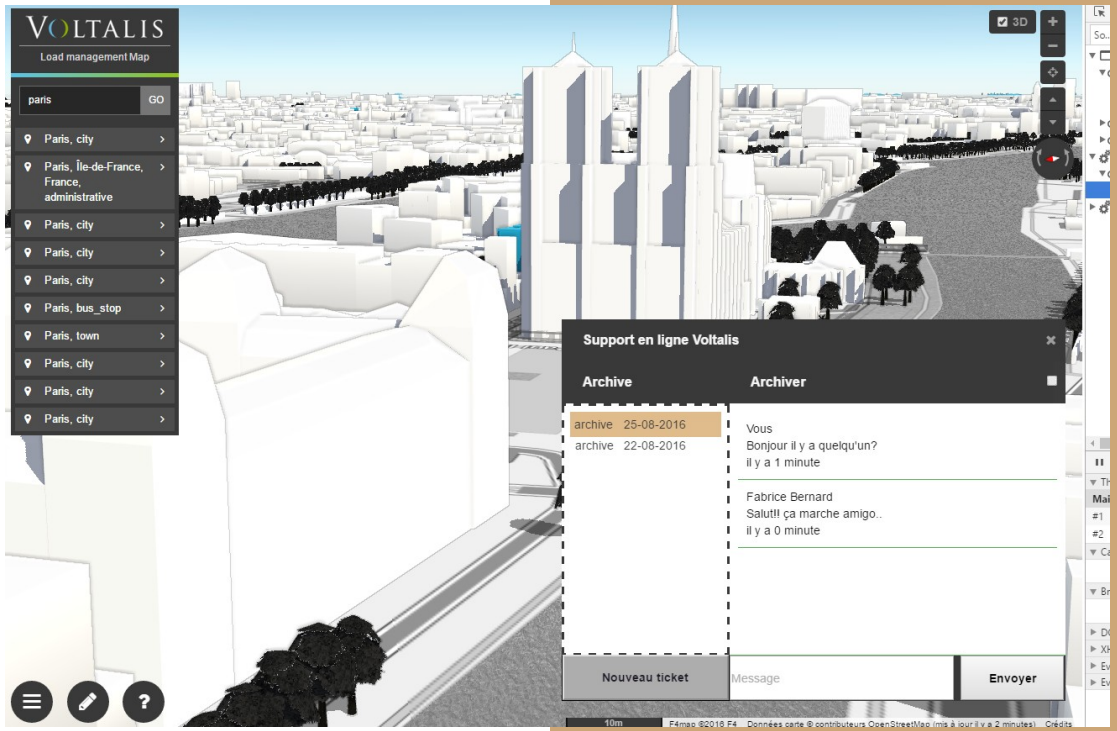
support client

Les maquettes permettent de **simuler l'utilisation** de la **future interface** de manière rapide en ignorant les problématiques de design.

Elles **valident le DAC**.



Frame contact support technique temps réel



Donc vue plus concrète

recherche

Extrait de code de l'index en Jade

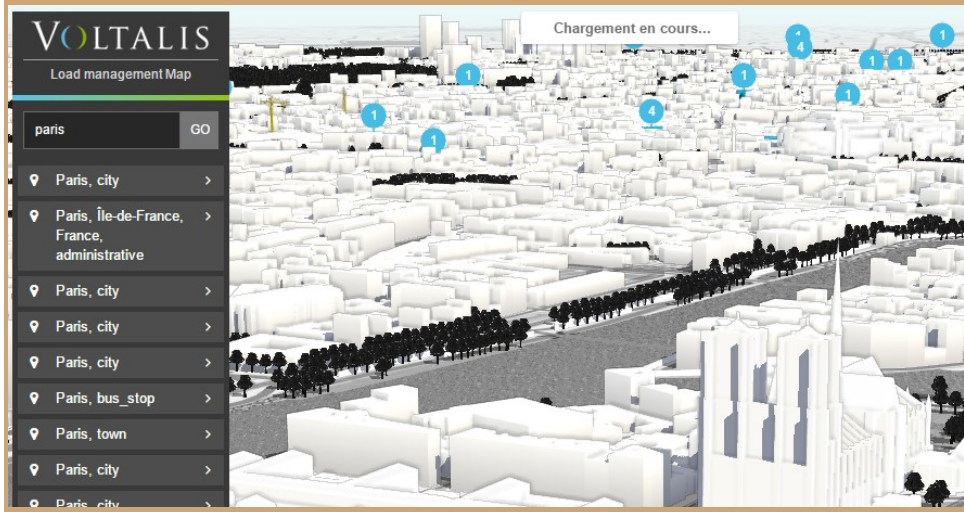
Ce morceau de code représente le **champs de saisie** utilisé par le **moteur de recherche** et l'élément dans lequel je l'ai inclus :

```
Header
    .heaterTitle
        img(src=resourceUrl("/images/logo.png"))
        .pitch
            |Load management Map
    .headerLegend
    .headerSearch
        input#search(type='text',
 placeholder='Rechercher')
        button#searchBtn GO
    .headerSearchResultContainer
```

Extrait de code, le template Jade contenant les résultats

Il est bon de noter la **fonction « mixin »** qui **définit un modèle** et permet de le **réutiliser** par exemple pour des **itérations dynamique depuis Javascript** :

```
mixin searchResultItem(result)
    .searchResultItem(data-id=result.osm_id)
        i.firstIcon.fa.fa-map-marker
        span.searchResultItem_title= result.name +
 ", " + result.nominatim.type
        i.lastIcon.fa.fa-angle-right

 .headerSearchResultContent
    each result in results.values
        +searchResultItem(result)
```

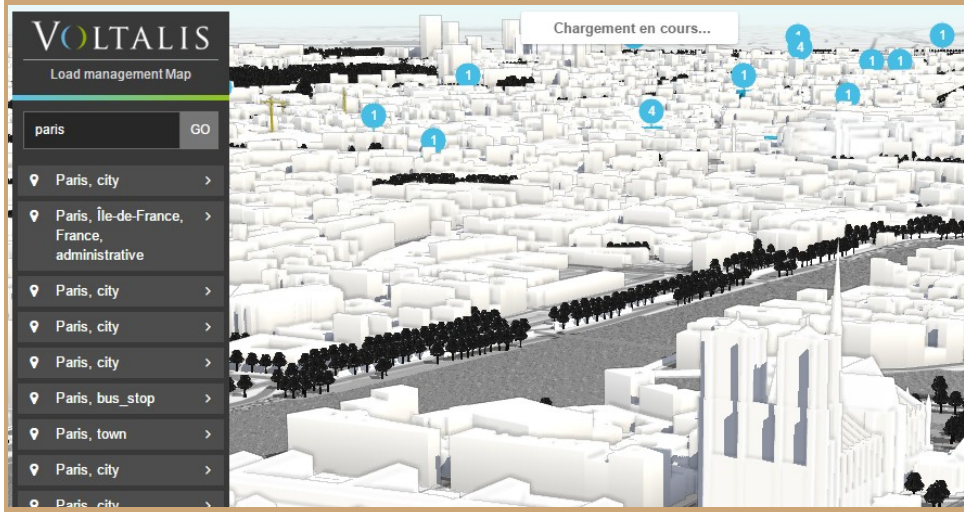


- Un champs de saisie avec bouton [GO]
- Liste de résultats sélectionnables par click
- Pilotage de la caméra sur 1er résultat ou sélection de l'utilisateur

recherche

Extrait de code Stylus, le CSS de la liste de résultats

```
.headerSearchResultContainer
 padding 5px 5px 0 5px
 .searchResultItem
   font-size 13px
   display table
   width 100%
   margin-bottom 5px
   background rgba(255,255,255,0.1)
   cursor pointer
   transition all ease 0.3s
   &:hover
     background rgba(255,255,255,0.3)
   .firstIcon
     width 30px
     text-align center
     display table-cell
   span.searchResultItem_title
     display table-cell
     padding 5px
   .lastIcon
     width 30px
     text-align center
     display table-cell
```



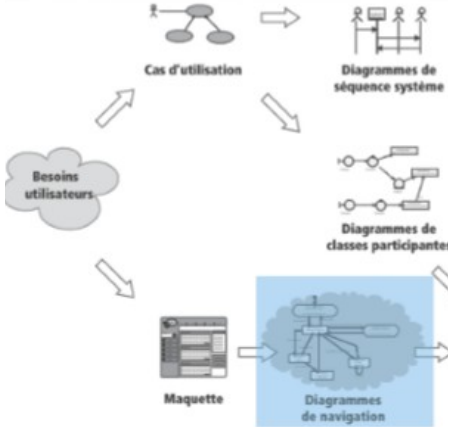
- Respect de la charte graphique de l'application
- Les résultats sont affiché dans une cellule
- Au survol de la souris, les items se ternissent

Les diagrammes de navigation

Après avoir **préparé des maquettes** et les avoir **fait valider** par le client.

Il faut **relier** les différentes **interfaces**.

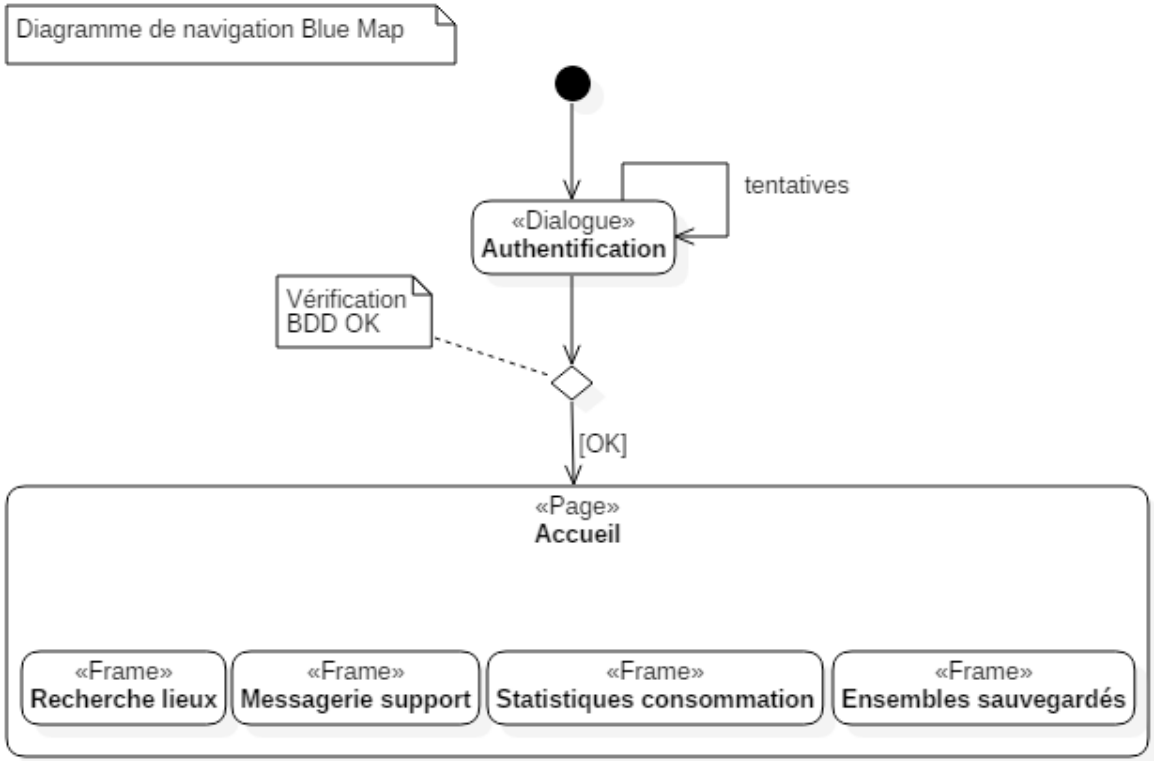
DNAV détermine **la logique** qui relie les fonctionnalités.



diagrammes de navigation

Les concepts importants :

- Point d’entrée
- Fenêtre, frame, dialogue
- États de l’application
- Transition
- condition
- Sortie ?

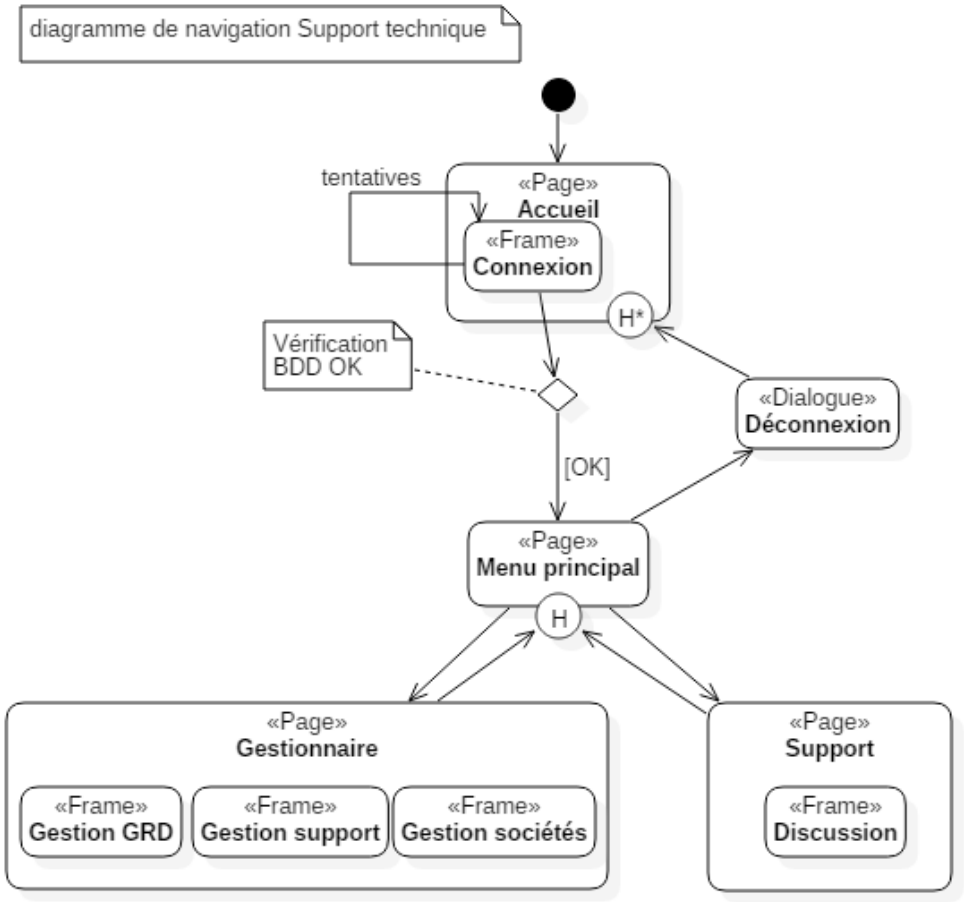


Blue Map

diagrammes de navigation

Les concepts importants :

- Point d’entrée
- Fenêtre, frame, dialogue
- États de l’application
- Transition
- condition
- Sortie ?



Support technique

Les diagrammes de séquence système

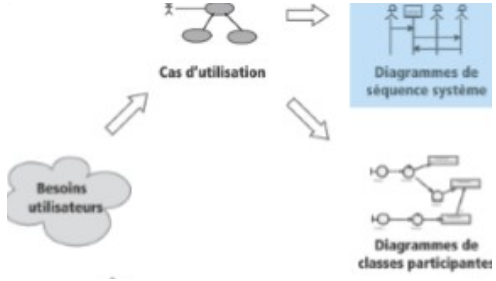
Après :

les cas d'utilisation

les maquettes

On spécifie **les messages** lors des **interactions** qui **appellent une réponse**.

C'est **en isolant ces messages** que nous allons représenter graphiquement des diagrammes d’interaction.

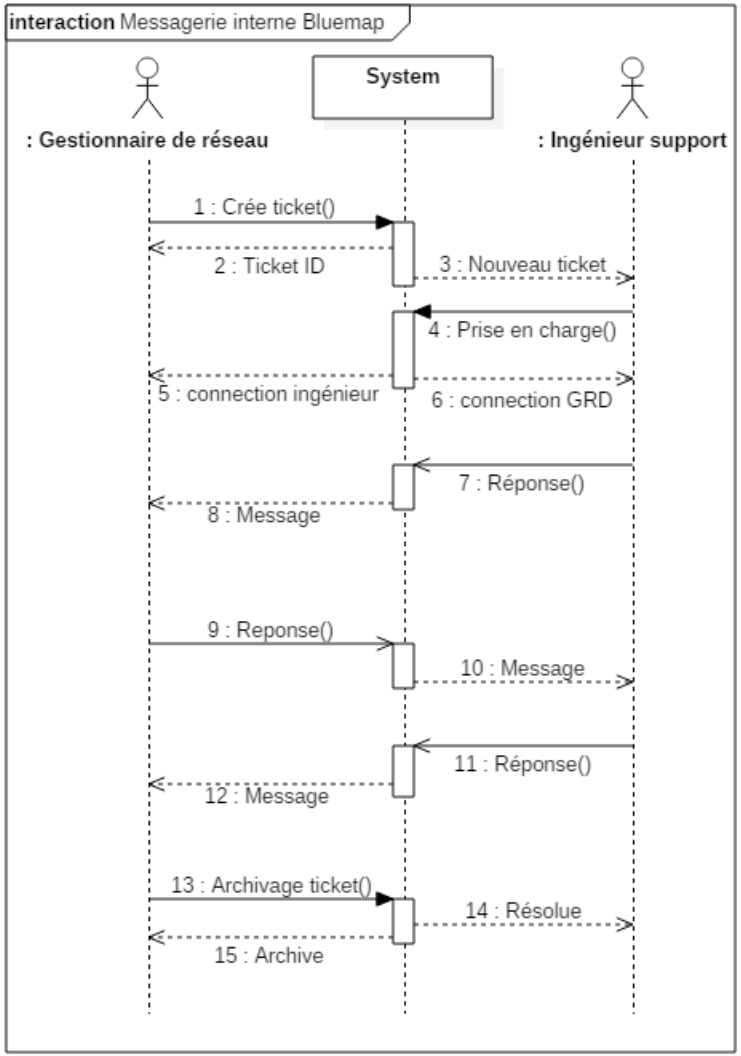


Diagrammes de séquence

Les concepts importants :

- Acteur
- Ligne de vie
- Requête
- Activation
- Réponse

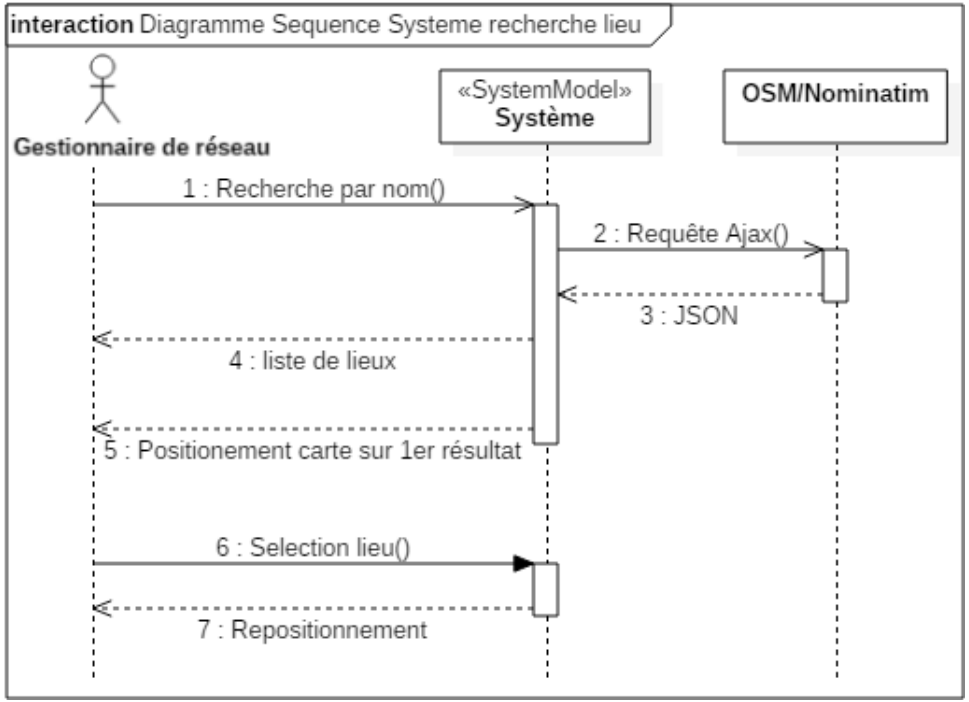
Support technique



Diagrammes de séquence

Les concepts importants :

- Acteur
- Ligne de vie
- Requête
- Activation
- Réponse

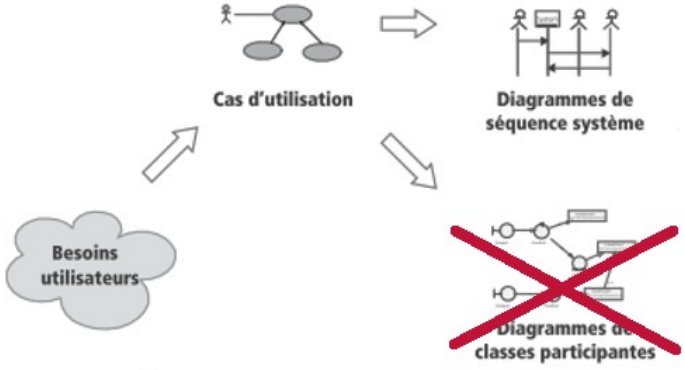


Moteur de recherche

Pas de classes en Javascript !

Je vais par contre représenter des instances d’objets.

Pour ça, un diagramme d’objet simplifié.



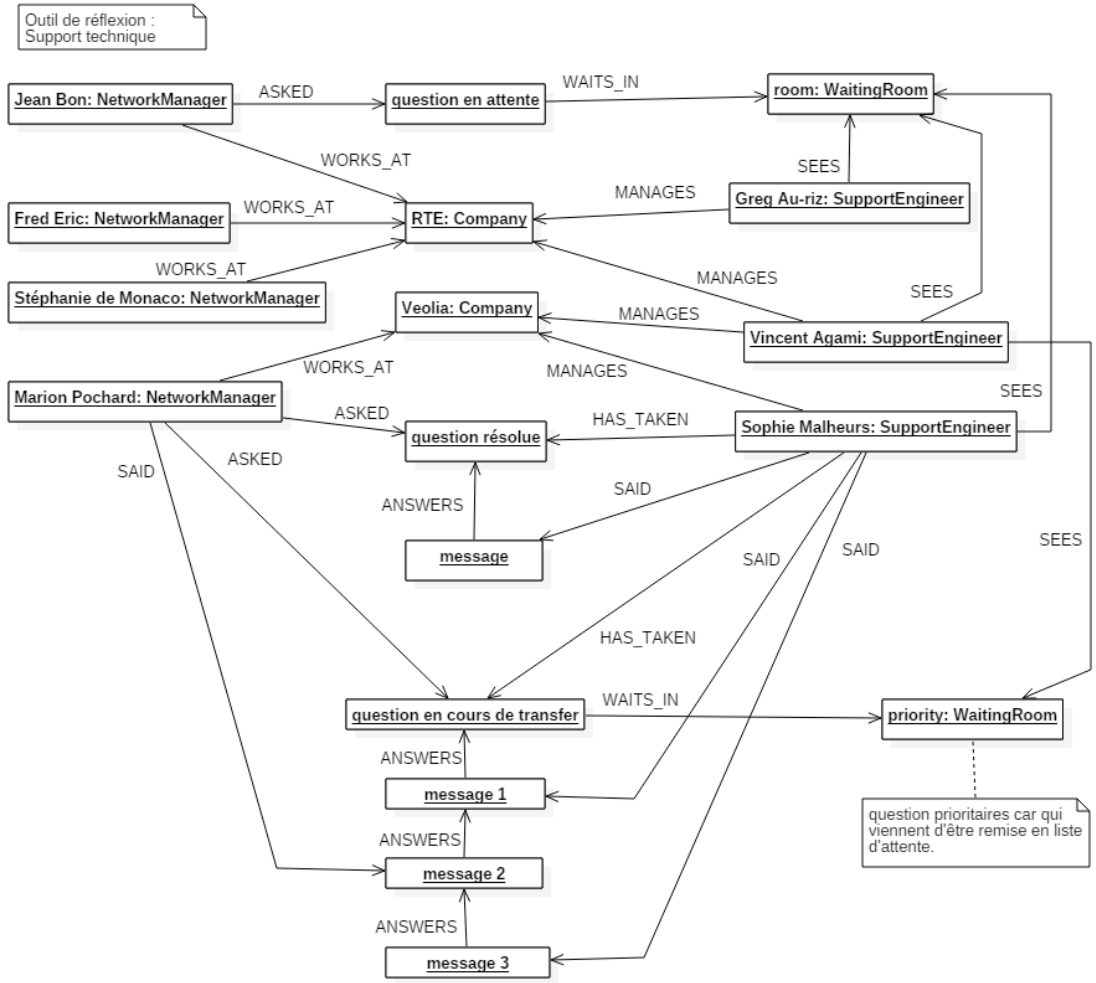
Diagrammes d’objets

Les concepts importants :

Objet conceptuel

Relation

Permet de **survoler un pseudo-état** rapidement lisible et mettre en lumière les questions de **règles de gestion importantes** mais **peu visibles.**

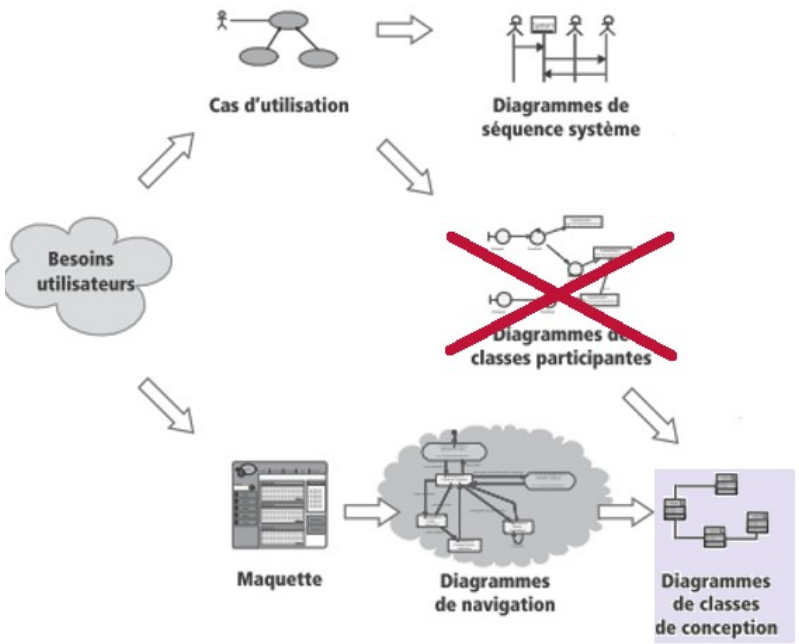


Support technique

Les diagrammes de classes

Identifiant les **concepts** du domaine, elles représentent les **entités significatives** de ce domaine.

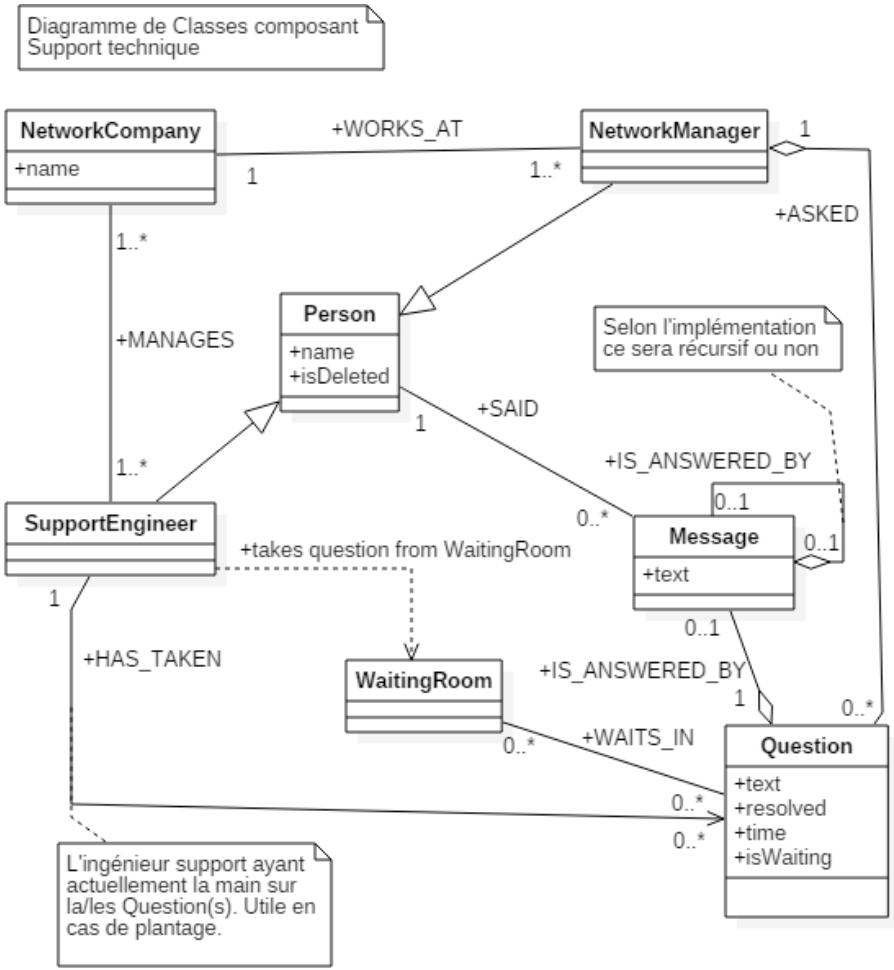
Il s’agit simplement de créer une **représentation visuelle** des **objets du monde réel dans un domaine donné**.



Diagrammes de classes

Les concepts importants :

- Classe conceptuelle
- Attribut
- Association
- Multiplicité
- Héritage
- Aggregation
- Composition



Support technique

relationnelles et graphe

SGBD-R/SQL	Graph/Neo4j/Cypher
Clé/élément, ligne, table, jointure, base de données.	Attribut, label, noeud, relation, cluster, instance de Neo4j.
Clé primaire, clé étrangère	Relations, pattern
Language impératif	Language déclaratif
Les **cardinalités** représentent le type de liens possible entre clés primaires, à un **niveau horizontal**.	Les **cardinalités** représentent la **profondeur de parcours** dans une **arborescence de relations** entre clusters.
La Force des SGBDR : **is unique**, **not null**, **indexed key** (une clé primaire est indexée implicitement).	Tout attribut peut : assert unique, assert not null, CREATE INDEX ON (un attribut unique est indexé implicitement).
Propriétés **ACID** (**atomicité**, **cohérence**, **isolation** et **durabilité**) qui **garantissent** qu'une **transaction informatique** est **exécutée de façon fiable**.	Propriétés **ACID** (**atomicité**, **cohérence**, **isolation** et **durabilité**) qui **garantissent** qu'une **transaction informatique** est **exécutée de façon fiable**.
Les **transactions**.	**Transactions disponible de manière programmatique** uniquement depuis Java via Java Transaction API (JTA)
Passé un **niveau de jointures et de volume données**, les SGBDR peine à répondre d'après certains benchmarks, mais **personnellement** je ne l'ai **jamais vu**!	Plusieurs millions de relations parcourues par seconde, **d'après certains benchmarks**, Neo4j continu de répondre dans des délais acceptables longtemps après

Langage déclaratif vs impératif

Les principales différences :

- Déclaratif
- Impératif
- Simplification de requête
- Parcours d'arborescence
- jointures

Exemple : Identifier tous les managers et compter leurs subordonnés sur 3 niveaux de profondeur

Cypher

```
MATCH (sub)-[:REPORTS_TO*0..3]->(boss),
      (report)-[:REPORTS_TO*1..3]->(sub)
WHERE boss.name = “John Doe”
RETURN sub.name AS Subordinate,
  count(report) AS Total
```

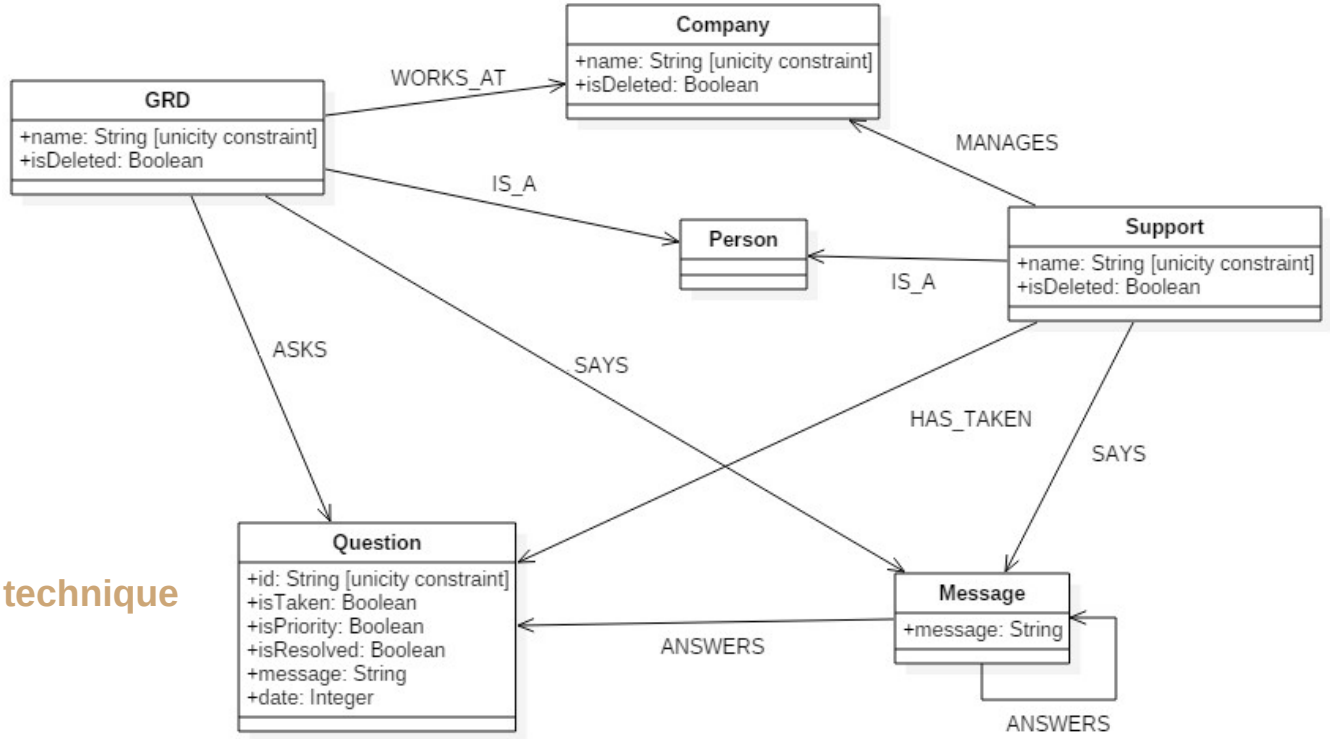
SQL

```
(SELECT T.directReportees AS directReportees, sum(T.count) AS count
FROM (
SELECT manager.pid AS directReportees, 0 AS count
  FROM person_reportee manager
  WHERE manager.pid = (SELECT id FROM person WHERE name = "fName lName")
UNION
  SELECT manager.pid AS directReportees, count(manager.directly_manages) AS count
FROM person_reportee manager
WHERE manager.pid = (SELECT id FROM person WHERE name = "fName lName")
GROUP BY directReportees
UNION
SELECT manager.pid AS directReportees, count(reportee.directly_manages) AS count
FROM person_reportee manager
JOIN person_reportee reportee
ON manager.directly_manages = reportee.pid
WHERE manager.pid = (SELECT id FROM person WHERE name = "fName lName")
GROUP BY directReportees
UNION
SELECT manager.pid AS directReportees, count(L2Reportees.directly_manages) AS count
FROM person_reportee manager
JOIN person_reportee L1Reportees
ON manager.directly_manages = L1Reportees.pid
JOIN person_reportee L2Reportees
ON L1Reportees.directly_manages = L2Reportees.pid
WHERE manager.pid = (SELECT id FROM person WHERE name = "fName lName")
GROUP BY directReportees
) AS T
GROUP BY directReportees)
UNION
(SELECT T.directReportees AS directReportees, sum(T.count) AS count
FROM (
SELECT manager.directly_manages AS directReportees, 0 AS count
FROM person_reportee manager
WHERE manager.pid = (SELECT id FROM person WHERE name = "fName lName")
UNION
SELECT reportee.pid AS directReportees, count(reportee.directly_manages) AS count
FROM person_reportee manager
JOIN person_reportee reportee
ON manager.directly_manages = reportee.pid
WHERE manager.pid = (SELECT id FROM person WHERE name = "fName lName")
GROUP BY directReportees
UNION

SELECT depth1Reportees.pid AS directReportees,
count(depth2Reportees.directly_manages) AS count
FROM person_reportee manager
JOIN person_reportee L1Reportees
ON manager.directly_manages = L1Reportees.pid
JOIN person_reportee L2Reportees
ON L1Reportees.directly_manages = L2Reportees.pid
WHERE manager.pid = (SELECT id FROM person WHERE name = "fName lName")
GROUP BY directReportees
) AS T
GROUP BY directReportees)
UNION
(SELECT T.directReportees AS directReportees, sum(T.count) AS count
  FROM(
  SELECT reportee.directly_manages AS directReportees, 0 AS count
FROM person_reportee manager
JOIN person_reportee reportee
ON manager.directly_manages = reportee.pid
WHERE manager.pid = (SELECT id FROM person WHERE name = "fName lName")
GROUP BY directReportees
UNION
SELECT L2Reportees.pid AS directReportees, count(L2Reportees.directly_manages)
AS count
FROM person_reportee manager
JOIN person_reportee L1Reportees
ON manager.directly_manages = L1Reportees.pid
JOIN person_reportee L2Reportees
ON L1Reportees.directly_manages = L2Reportees.pid
WHERE manager.pid = (SELECT id FROM person WHERE name = "fName lName")
GROUP BY directReportees
) AS T
GROUP BY directReportees)
UNION
(SELECT T.directReportees AS directReportees, sum(T.count) AS count
FROM (
SELECT manager.directly_manages AS directReportees, 0 AS count
FROM person_reportee manager
JOIN person_reportee L1Reportees
ON manager.directly_manages = L1Reportees.pid
JOIN person_reportee L2Reportees
ON L1Reportees.directly_manages = L2Reportees.pid
WHERE manager.pid = (SELECT id FROM person WHERE name = "fName lName")
)
```

Modèle physique des données(MPD)

Il représente **Une implémentation physique** dans une **base de données spécifique**, Neo4j dans mon cas.

- Noeud
- Label
- Attribut
- Relation
- Pattern

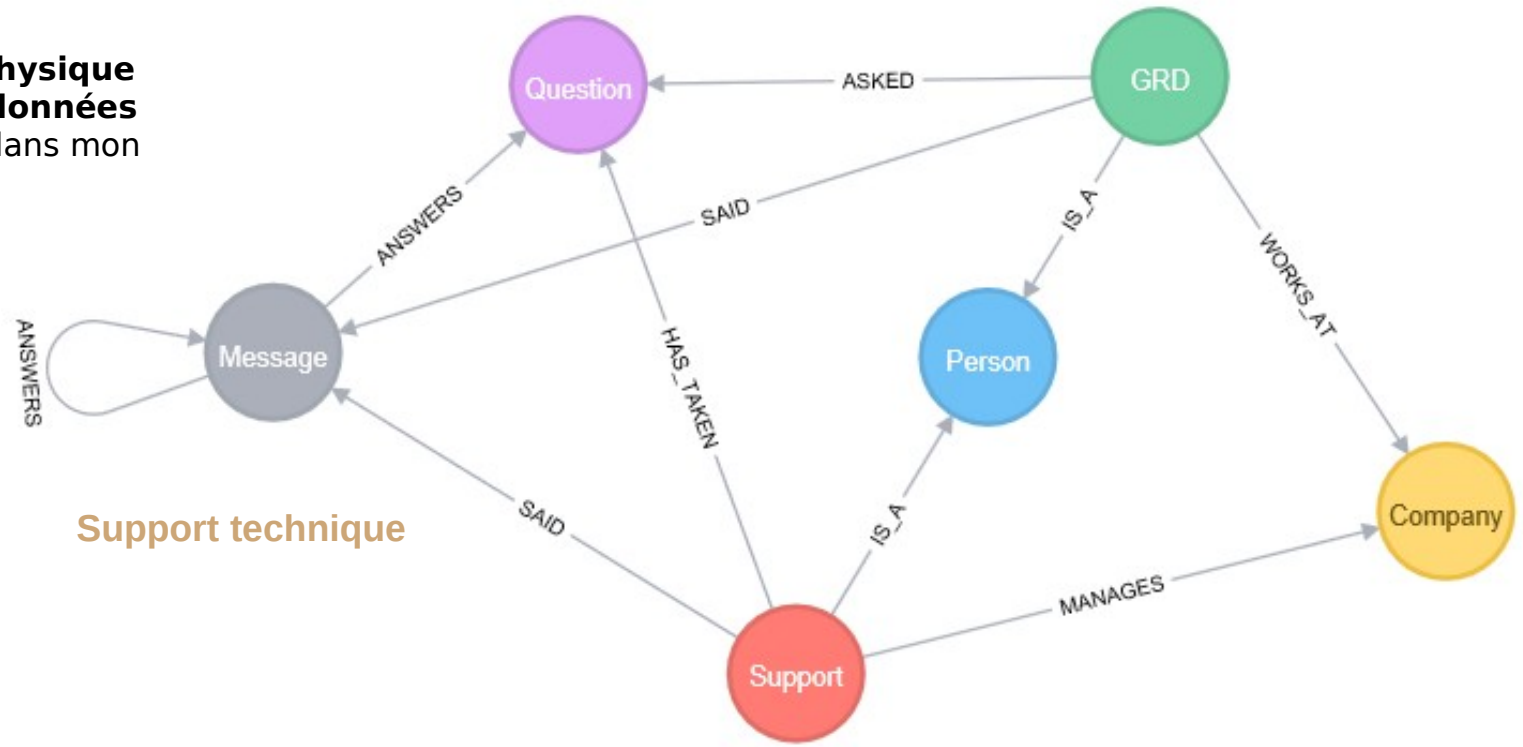


Support technique

Modèle physique des données(MPD)

Il représente **Une implémentation physique** dans une **base de données spécifique**, Neo4j dans mon cas.

- Noeud
- Label
- Attribut
- Relation
- Pattern



Création de la requête

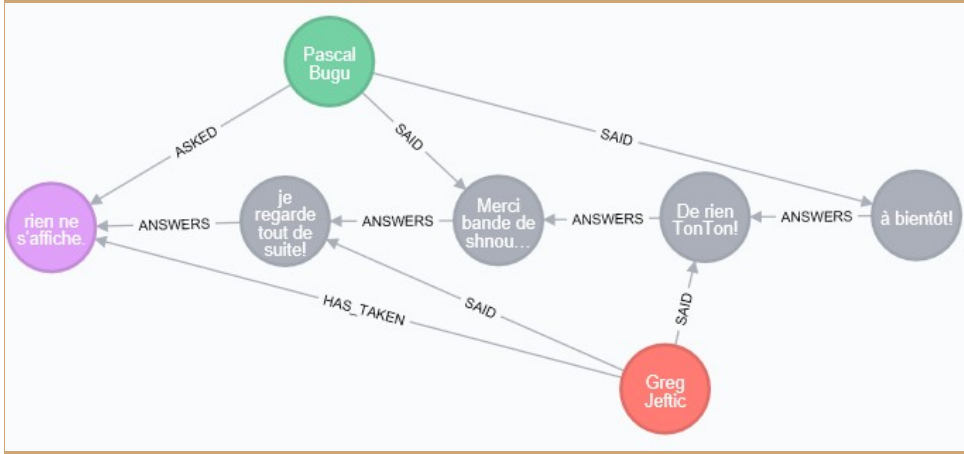
Requete cypher

```
MATCH (m)-[:SAID]->(a:Message)-
[:ANSWERS*0..]->(q:Question)<-
[:ASKED]-(p)
WHERE q.id = '57acfacb-34f6-4e0f-94d5-
c72601541667'
RETURN m,p,q,a
```

Requête préparée en Javascript

```
db.CypherQuery("MATCH (m)-[:SAID]->(a:Message)-[:ANSWERS*0..]-
>(q:Question)<-[:ASKED]-(p)
WHERE q.id = {id}
RETURN m,p,q,a", {
      id: '57acfacb-34f6-4e0f-94d5-c72601541667'
  }, function (err, resp) {
       if (err) {console.log(err);} else {console.log(resp)}
      });
```

Visualisation de la requête



Extrait de code permettant de repeupler la base de données, extrait depuis le “shell” de Neo4j via une procédure stockée

Les blocs de transaction permettant de garantir le bon déroulement de la restauration de la base, et ne pas surcharger la Java virtual machine

```
begin    (ici, on démarre un bloc de transaction)
        CREATE (:`Support`:`UNIQUE IMPORT LABEL` {`name`:"Greg Jeftic", `role`:"support", `UNIQUE IMPORT ID`:0});
        CREATE (:`Support`:`UNIQUE IMPORT LABEL` {`name`:"Ludo Peper", `role`:"support", `UNIQUE IMPORT ID`:1});
        CREATE (:`Support`:`UNIQUE IMPORT LABEL` {`name`:"Fab Beber", `role`:"support", `UNIQUE IMPORT ID`:2});
        CREATE (:`UNIQUE IMPORT LABEL` {`name`:"Mathieu Bineau", `UNIQUE IMPORT ID`:3});
        CREATE (:`Company` {`name`:"RTE"});
        CREATE (:`GRD`:`UNIQUE IMPORT LABEL` {`name`:"Pascal Bugu", `UNIQUE IMPORT ID`:7});
        CREATE (:`Question` {`id`:"57acfacb-34f6-4e0f-94d5-c72601541667", `message`:"rien ne s'affiche.", `resolved`:false});
        CREATE (:`Message`:`UNIQUE IMPORT LABEL` {`message`:"je regarde tout de suite!", `UNIQUE IMPORT ID`:9});
        CREATE (:`Person`:`UNIQUE IMPORT LABEL` {`is`:"human", `UNIQUE IMPORT ID`:10});
        CREATE (:`Message`:`UNIQUE IMPORT LABEL` {`message`:"Merci bande de shnoufs!", `UNIQUE IMPORT ID`:11});
        CREATE (:`Company` {`name`:"ERDF"});
        CREATE (:`GRD`:`UNIQUE IMPORT LABEL` {`name`:"Jean Bon", `UNIQUE IMPORT ID`:24});
        commit  (on indique à la base d'exécuter le bloc si tout se passe bien)
        begin
        CREATE CONSTRAINT ON (node:`Question`) ASSERT node.`id` IS UNIQUE;
        CREATE CONSTRAINT ON (node:`Person`) ASSERT node.`name` IS UNIQUE;
        CREATE CONSTRAINT ON (node:`Company`) ASSERT node.`name` IS UNIQUE;
        CREATE CONSTRAINT ON (node:`Pool`) ASSERT node.`access` IS UNIQUE;
        CREATE CONSTRAINT ON (node:`UNIQUE IMPORT LABEL`) ASSERT node.`UNIQUE IMPORT ID` IS UNIQUE;
        commit
        schema await    (cet instruction est importante! Faire patienter jusqu’à la fin des transaction précédentes avant de transmettre les blocs suivants!)
```

Extrait de code permettant de repeupler la base de données, extrait depuis le “shell” de Neo4j via une procédure stockée

Les blocs de transaction permettant de garantir le bon déroulement de la restauration de la base, et ne pas surcharger la Java virtual machine

begin

On ancre des noeuds puis on reconstruit les relations entre les noeuds “n1” et “n2” par ligne

```
MATCH (n1:`UNIQUE IMPORT LABEL`{`UNIQUE IMPORT ID`:3}), (n2:`Company`{`name`:"Voltalis"}) CREATE (n1)-[:`WORKS_AT`]->(n2);
MATCH (n1:`UNIQUE IMPORT LABEL`{`UNIQUE IMPORT ID`:7}), (n2:`Company`{`name`:"RTE"}) CREATE (n1)-[:`WORKS_AT`]->(n2);
MATCH (n1:`UNIQUE IMPORT LABEL`{`UNIQUE IMPORT ID`:7}), (n2:`Question`{`id`:"57acfacb-34f6-4e0f-94d5-c72601541667"}) CREATE (n1)-[:`ASKED`
  {`date`:1475069322618}]->(n2);
MATCH (n1:`UNIQUE IMPORT LABEL`{`UNIQUE IMPORT ID`:0}), (n2:`UNIQUE IMPORT LABEL`{`UNIQUE IMPORT ID`:9}) CREATE (n1)-[:`SAID`
  {`date`:1475070593324}]->(n2);
MATCH (n1:`UNIQUE IMPORT LABEL`{`UNIQUE IMPORT ID`:9}), (n2:`Question`{`id`:"57acfacb-34f6-4e0f-94d5-c72601541667"}) CREATE (n1)-[:`ANSWERS`
  {`date`:1475070593324}]->(n2);
MATCH (n1:`UNIQUE IMPORT LABEL`{`UNIQUE IMPORT ID`:1}), (n2:`Company`{`name`:"RTE"}) CREATE (n1)-[:`MANAGES`]->(n2);
MATCH (n1:`UNIQUE IMPORT LABEL`{`UNIQUE IMPORT ID`:7}), (n2:`UNIQUE IMPORT LABEL`{`UNIQUE IMPORT ID`:11}) CREATE (n1)-[:`SAID`
  {`date`:1475770481223}]->(n2);
MATCH (n1:`UNIQUE IMPORT LABEL`{`UNIQUE IMPORT ID`:11}), (n2:`UNIQUE IMPORT LABEL`{`UNIQUE IMPORT ID`:9}) CREATE (n1)-[:`ANSWERS`]->(n2);
MATCH (n1:`UNIQUE IMPORT LABEL`{`UNIQUE IMPORT ID`:0}), (n2:`UNIQUE IMPORT LABEL`{`UNIQUE IMPORT ID`:12}) CREATE (n1)-[:`SAID`
  {`date`:1475770696896}]->(n2);
MATCH (n1:`UNIQUE IMPORT LABEL`{`UNIQUE IMPORT ID`:24}), (n2:`Company`{`name`:"ERDF"}) CREATE (n1)-[:`WORKS_AT`]->(n2);
MATCH (n1:`UNIQUE IMPORT LABEL`{`UNIQUE IMPORT ID`:0}), (n2:`Company`{`name`:"ERDF"}) CREATE (n1)-[:`MANAGES`]->(n2);
MATCH (n1:`UNIQUE IMPORT LABEL`{`UNIQUE IMPORT ID`:1}), (n2:`Company`{`name`:"ERDF"}) CREATE (n1)-[:`MANAGES`]->(n2);
MATCH (n1:`UNIQUE IMPORT LABEL`{`UNIQUE IMPORT ID`:2}), (n2:`Company`{`name`:"ERDF"}) CREATE (n1)-[:`MANAGES`]->(n2);
MATCH (n1:`UNIQUE IMPORT LABEL`{`UNIQUE IMPORT ID`:0}), (n2:`Company`{`name`:"RTE"}) CREATE (n1)-[:`MANAGES`]->(n2);
MATCH (n1:`UNIQUE IMPORT LABEL`{`UNIQUE IMPORT ID`:12}), (n2:`UNIQUE IMPORT LABEL`{`UNIQUE IMPORT ID`:11}) CREATE (n1)-[:`ANSWERS`]->(n2);
MATCH (n1:`UNIQUE IMPORT LABEL`{`UNIQUE IMPORT ID`:7}), (n2:`UNIQUE IMPORT LABEL`{`UNIQUE IMPORT ID`:10}) CREATE (n1)-[:`IS_A`]->(n2);
MATCH (n1:`UNIQUE IMPORT LABEL`{`UNIQUE IMPORT ID`:24}), (n2:`UNIQUE IMPORT LABEL`{`UNIQUE IMPORT ID`:10}) CREATE (n1)-[:`IS_A`]->(n2);
```

Les diagrammes de séquence détaillée

Différentes couches métier du système, représente un **ensemble d'objets en interaction**. Deux objets **utilisateurs** du système et des objets du **système** : pages, boîtes de dialogue (**view**), contrôleurs (**controller**) et entités (**model**) interagissant entre eux.

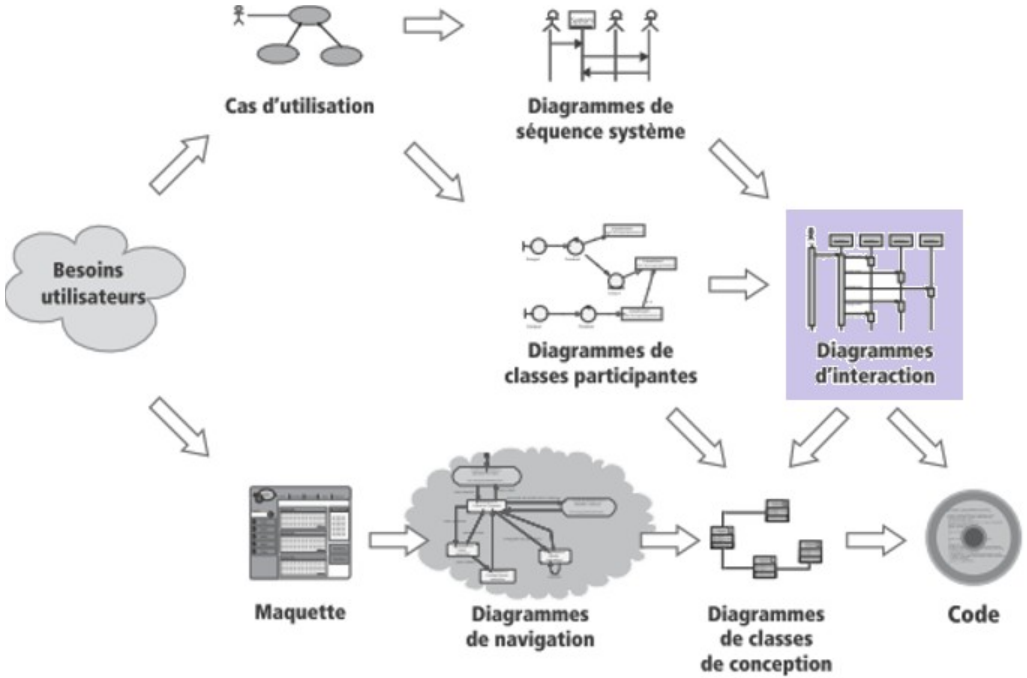


Figure 1–20 Schéma complet du processus de modélisation d'une application web

Diagrammes de séquence détaillée

Les concepts importants :

Acteur

Boundaries

Contrôleur

Entités

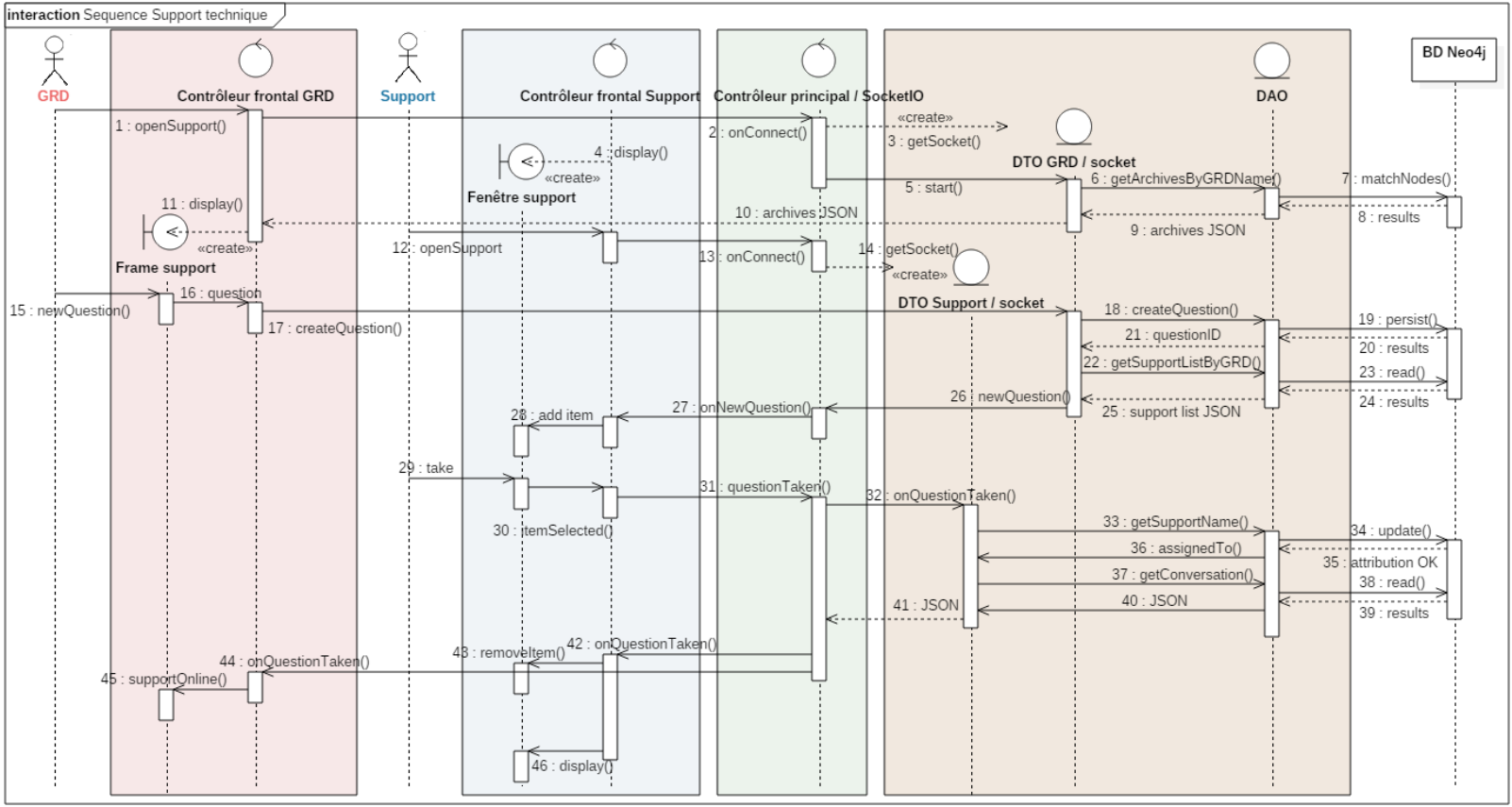
BDD

Message

Création

Asynchrone

Support technique



Comparatif entre langages de classes et Javascript

Langages de classes

Classes et instances sont **deux entités distinctes**.

Une **classe est définie** avec une définition de classe. On **instancie une classe** avec des **constructeurs**

On crée **un seul objet** grâce à l'**opérateur new**.

On construit une **hiérarchie d'objets** en utilisant les **définitions des classes** pour **définir des classes-filles** à partir de **classes existantes**.

Les **objets héritent** des propriétés appartenant à la **chaîne des classes de la hiérarchie**

Impossible d'ajouter des propriétés dynamiquement pendant l'exécution.

Javascript

Tous les objets sont des instances.

On définit et on crée un ensemble d'objets avec des fonctions qui sont des **constructeurs** ou de **manière “littérale”**

Pareil.

On construit une hiérarchie d'objets en **assignant un prototype à un objet** dans le **constructeur de cet objet**.

Les objets héritent des propriétés appartenant à la **chaîne des prototypes de la hiérarchie**.

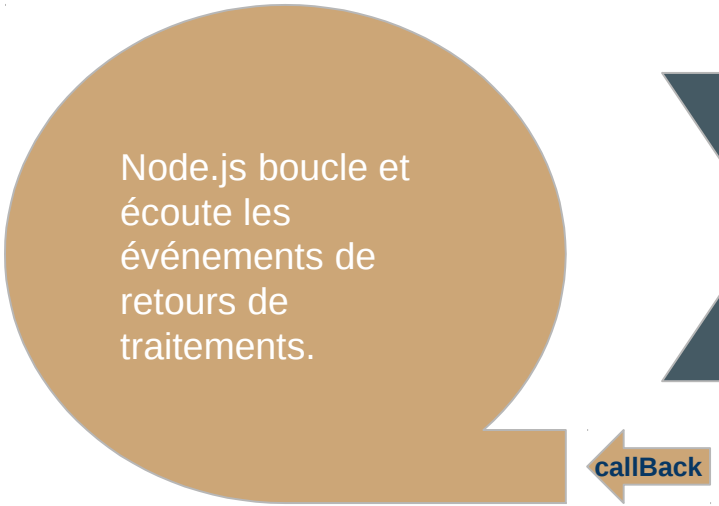
Le constructeur ou le prototype définit un ensemble de propriétés **initiales**. Il est **possible d'ajouter** ou **de retirer des propriétés dynamiquement** de manière “**littérale**”.

événementielle

événementielle

Les workers Node.js créent des threads qui bouclent en attente d’un “event”

Dès réception d’une réponse, la callBack s’active



Lorsqu’un traitement externe via une bibliothèque asynchrone est achevé, le résultat est passé en argument de la fonction callBack qui traite alors sa portée.

Exemple DTO

```
var GRD = {
   toJSON: function(obj){
       return {
       name: obj.name,
       newName: obj.newName,
       company: obj.company,
       isDeleted: obj.isDeleted};
   }
};
```

L'appel de la méthode getGRDByName de l'objet d'accès aux données DAO renvoie un résultat sous forme JSON.

```
GRD = DAO.getGRDByName("Jean Bon");
```

On pourra alors le sérialiser afin de le transférer entre serveur et client.

```
JSON.stringify(GRD);
```

Exemple DAO

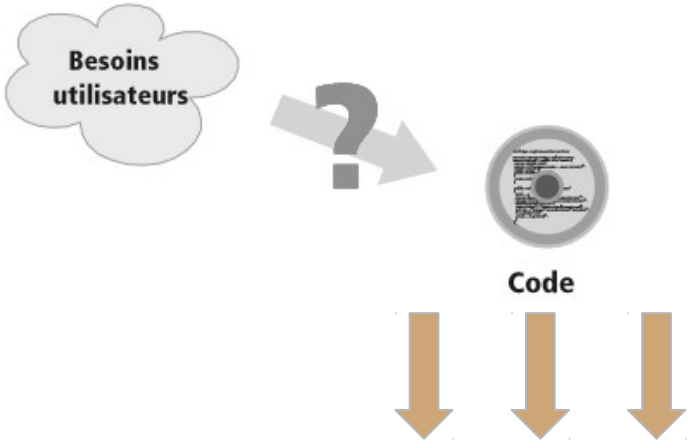
```
// Require the Neo4J module drivers
var neo4j = require('node-neo4j');

// Create a db object. We will using this object to work on
the DB.
var db = new
neo4j('http://username:password@localhost:7474');
var DAO = new Object();


DAO = {
 getMessageByQuestionId: function (id){
   db.cypherQuery("MATCH (m)-[:ANSWERS]->(q:Question)"
                  + "where q.id = {questionID}"
                  + "return m AS message",
                  {questionID: id},
                  function (err, result) {
                      if (err) {
                          return console.log(err);
                      }
                      return result.data;
                  });
   }
}
```

Les diagrammes de déploiement

Une fois l'**application terminée**, il faut la rendre accessible aux **utilisateurs**.

On parle alors **déploiement** ou de **mise en production**.

Les multiplicités, indique s'il y a besoin d'**optimisations** entre noeuds!



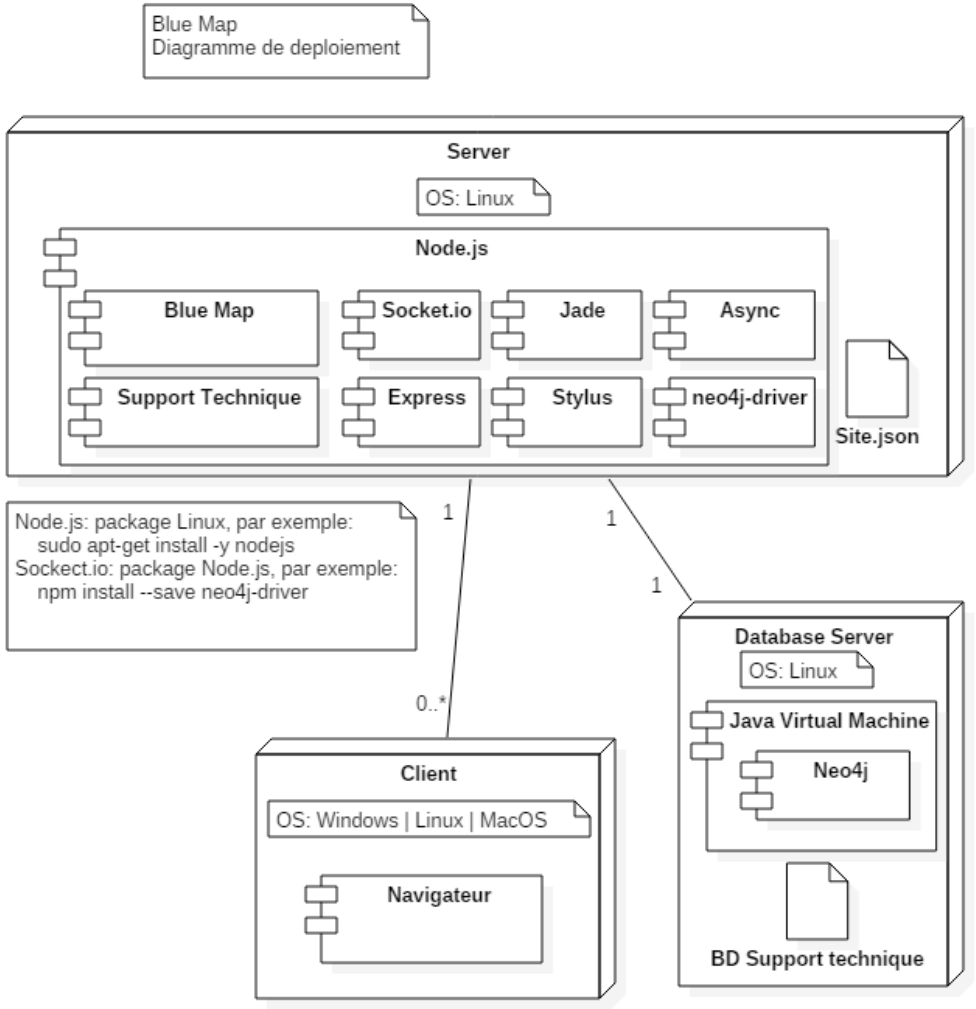
Serveur,
Serveur de base de données,
Postes client.

Diagrammes de déploiement

Les concepts importants :

- Noeuds
- Composants
- Artefacts
- Association
- Multiplicité

Application Blue Map



Les technologies utilisées

Environnement local

Front-end

Back-end

Things that matter

Faire de son mieux avec un temps imparti

Savoir être autonome

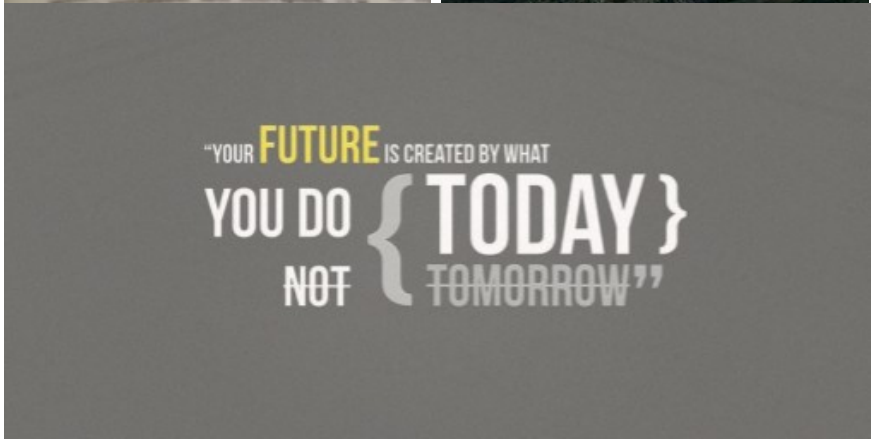
Lire une doc, chercher l’info où elle se trouve

Persévérer, mais savoir prendre son temps

Ne pas oublier la passion

La méthode pomodori

Penser « u



Pour finir

… je suis passé par pas mal d’étapes !