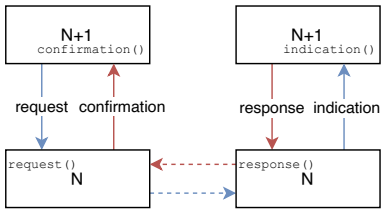
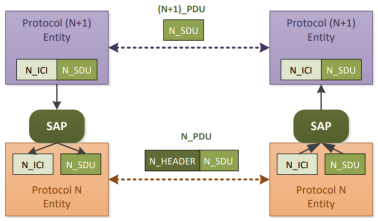
1 Introduction

1.1 OSI





Un SAPI permet de choisir quel protocole utiliser (si il y a plusieurs protocoles)

2 C++

2.1 friend

```
class A {
private:
  // La classe B peut accéder aux méthodes
    privées de A
  friend class B;
  // La fonction calc de C peut accédéer aux
    méthodes privées de A
  friend int C::calc(int x);
  // La fonction main peut accédéer aux mé
    thodes privées de A (à éviter)
  friend int main();
}
```

2.2 Polymorphisme

2.2.1 static binding

```
class A {
public:
  void display() { cout << "A" << endl;};
};
class B : public A {
public:
  void display() {cout << "B" << endl;};
};

int main() {
  A a;
  B b;
  A* p;
  p = &a;
  p->display(); -> "A"
  p = &b;
  p->display(); -> "A"
}
```

2.2.2 Dynamic binding

```
class A {
public:
  virtual void display() { cout << "A" <<
    endl;};
};
class B : public A {
public:
  virtual void display() {cout << "B" << endl
    ;};
};

int main() {
  A a;
  B b;
  A* p;
  p = &a;
  p->display(); -> "A"
  p = &b;
  p->display(); -> "B"
}
```

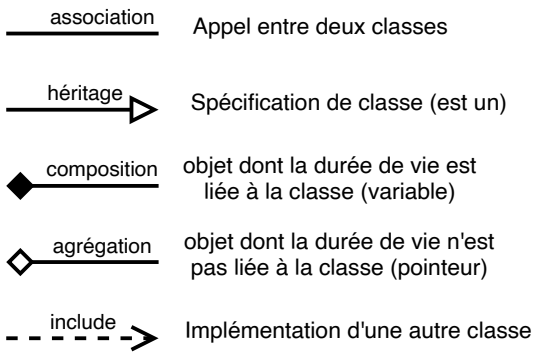
2.2.3 Interfaces

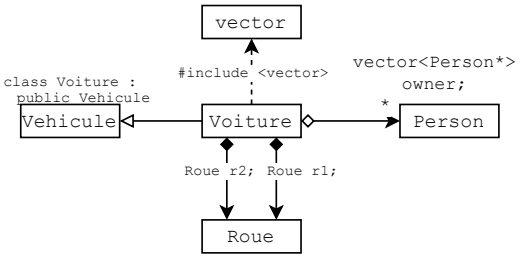
```
class IVehicle {
public:
  virtual void drive() = 0;
};
class Car : public IVehicle {
public:
  virtual void drive() { cout << "car drives"
     << endl;}
};
class Rocket : public IVehicle {
public:
  virtual void drive() { cout << "rocket
    flies" << endl;}
};
int main() {
  IVehicle* v1 = new Rocket();
  IVehicle* v2 = new Car();
  v1->drive(); // car
  v2->drive(); // rocket
  delete v1;
  delete v2;
};
```

2.3 Classes génériques

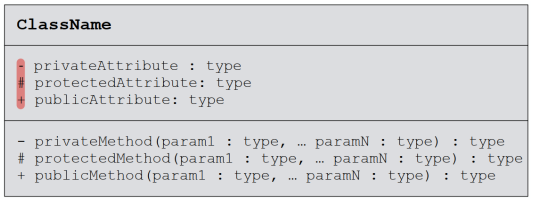
```
vector<T> vInt(5);
```

2.4 UML

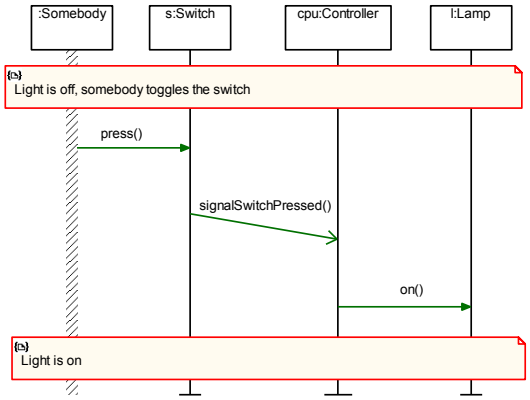




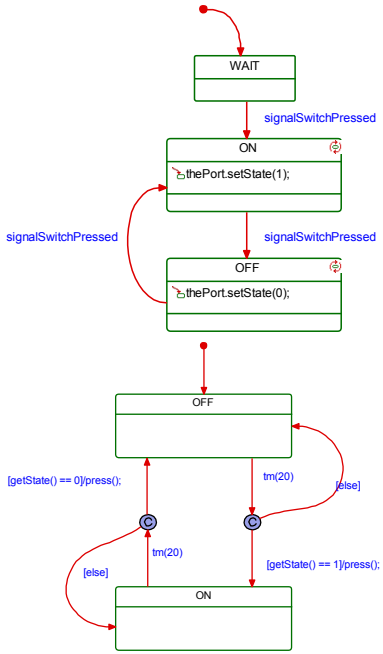
2.4.1 Diagrammes de classe



2.4.2 Diagrammes de séquences



2.4.3 Diagrammes d'états



2.5 Singleton

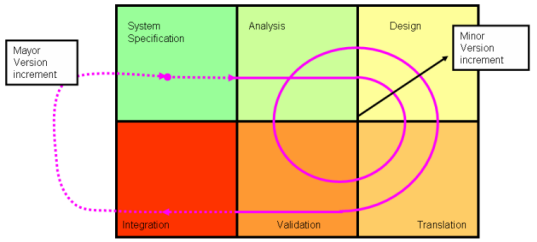
```
class Singleton {
public:
  static Singleton& getInstance() {
    static Singleton instance;
    return instance;
  }
private:
  // Private constructor and desctructor
  Singleton() {};
  Singleton(const Singleton&) {};
  void operator=(const Singleton&) {};
};

int main() {
  Singleton::getInstace().doSomething();
}
```

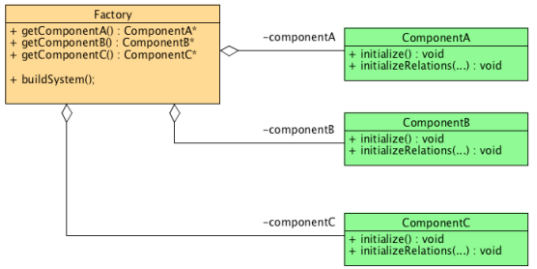
3 Patterns

3.1 Process

3.1.1 6Q



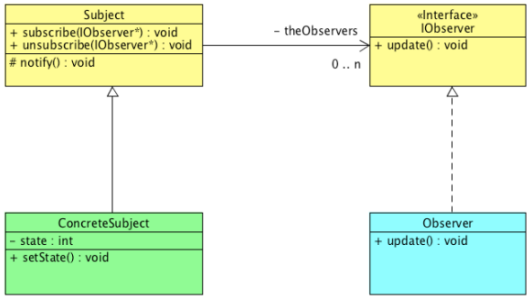
3.2 Factory



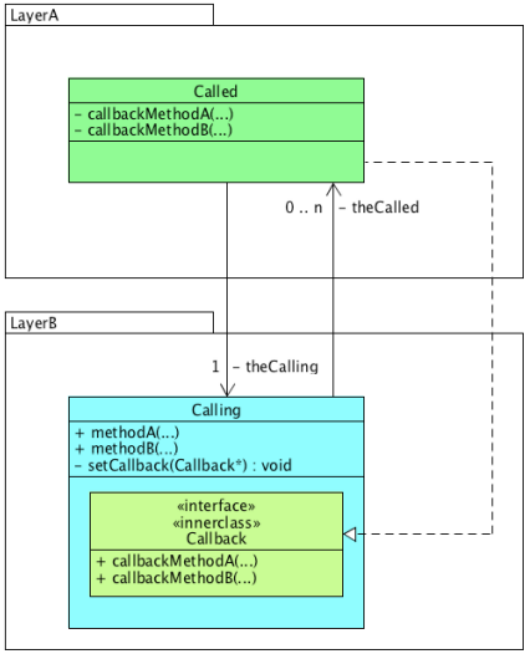
Dans `buildSystem()` :

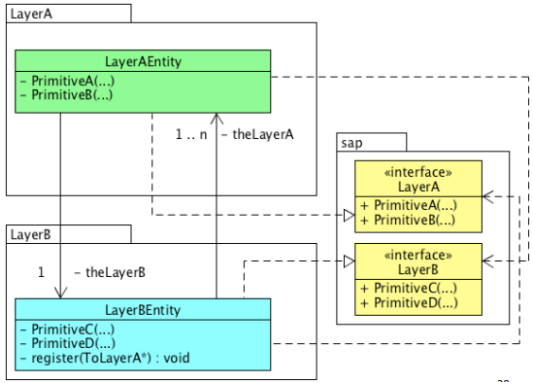
1. `create`
2. `initialize`
3. `initializeRelations` quand tous les `create` et `initialize` sont faits

3.3 Observateur



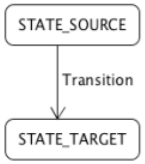
3.4 Interface

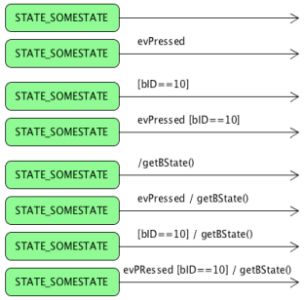




4 XF et Machines d'états

1. États
   (a) Action sur entrée
   (b) Action continue
   (c) Action sur sortie
2. Transistions
   (a) déclenchée par un événement
   (b) Condition
   (c) Action



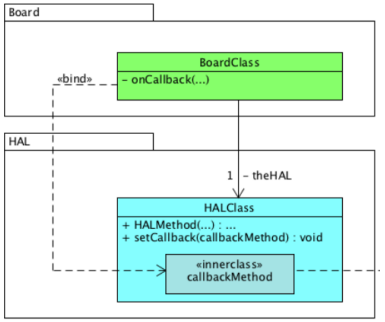


4.1 XF

Atomique : Pas d'interruptions

5 HAL

1. Portabilité
2. Réutilisable



N	Nom	unité	description	exemples
7	Application	Données	Utilité pour l'utilisateur (transfert de fichiers, vidéos, etc...)	Web, FTP, IMAP, LDAP, HTTP, SMB
6	Présentation	Données	Formats, mises en formes, cryptage, login	JSON, ASCII, HTML, Unicode
5	Session	Données	Gestion de l'activité	RPC, NetBios
4	Transport	Segments, streams	sous-adressage, communication entre deux processus	TCP, UDP
3	Réseau	Packets	Transport des données dans un réseau maillé	IPv4/IPv6, ARP
2	Liaison	Trame	Adressage local, gestion des erreurs, etc...	Ethernet, CAN,
1	Physique	Bit	Signaux électriques	Wi-Fi, Câble, 1000BASE-T, USB

Les couches 1-4 permettent de transférer les données. Les couches 5-7 sont liées à l'utilisation qu'on fait des données.