1 Modèle OSI et concepts liés

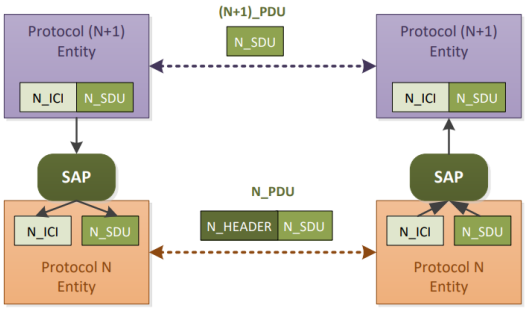
— ISO : International Organization for Standardization ($\xrightarrow{\text{create}}$ OSI)

— OSI : Open Systems Interconnection

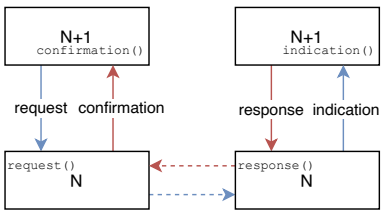
— SDU : Service Data Unit

— PDU : Protocol Data Unit

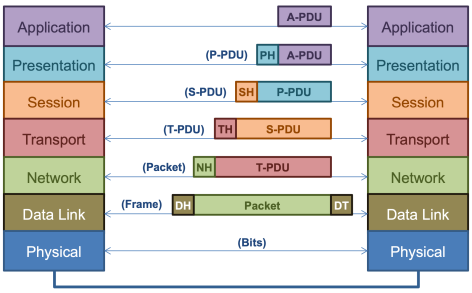
« Le modèle OSI ne spécifie pas un protocole. Il s'agit plutôt d'un framework pour la spécification de protocole. »
OSI introduit la notion de **service** (**SAP** - Service Access Point) avec un Service User et un Service Provider.



Basé sur **4 primitives** de service.



Modèle OSI complet :



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

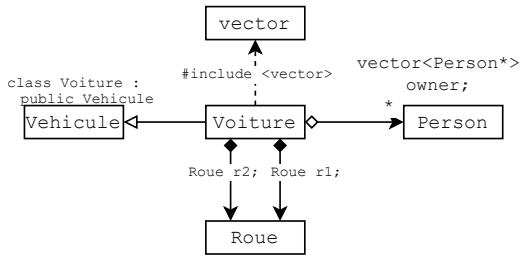
```
class IVehicle {
public:
  virtual void drive() = 0;
};
class Car : public IVehicle {
public:
  virtual void drive() { cout << "car drives
    " << endl;}
};
class Rocket : public IVehicle {
public:
  virtual void drive() { cout << "rocket
    flies" << endl;}
```

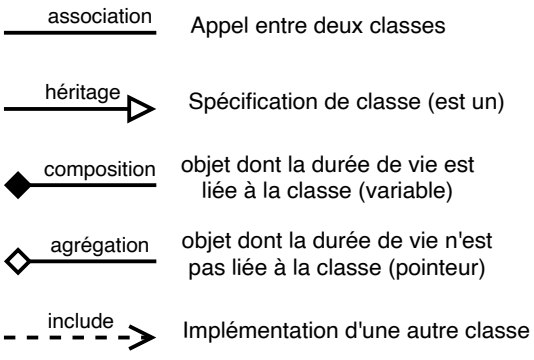
```
};
int main() {
  IVehicle* v1 = new Rocket();
  IVehicle* v2 = new Car();
  v1->drive(); // car
  v2->drive(); // rocket
  delete v1;
  delete v2;
};
```

2.3 Classes génériques

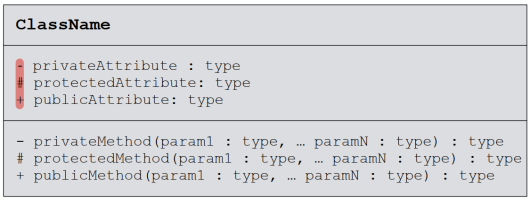
```
vector<T> vInt(5);
```

2.4 UML

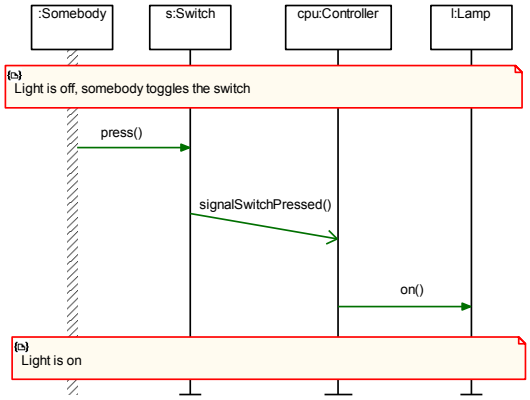




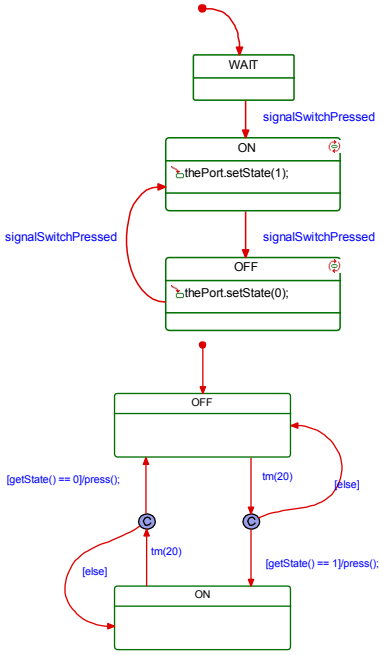
2.4.1 Diagrammes de classe



2.4.2 Diagrammes de séquences

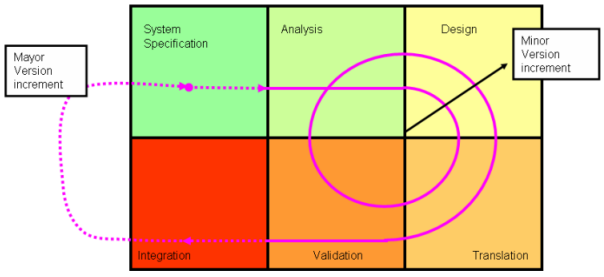


2.4.3 Diagrammes d'états



3 Patterns

3.1 6Q Process



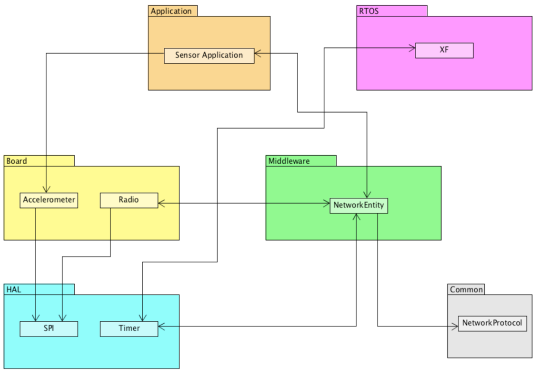
3.2 Singleton

```
class Singleton {
public:
  static Singleton& getInstance() {
    static Singleton instance;
    return instance;
```

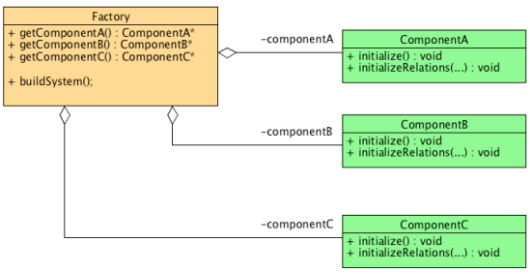
```
}
private:
  // Private constructor and desctructor
  Singleton() {};
  Singleton(const Singleton&) {};
  void operator=(const Singleton&) {};
};

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

3.3 Concept of 5 layers

1. **Application :** contrôle de l'application
2. **Middleware :** éléments de communication
3. **RTOS :** OS temps réel
4. **Board :** mise en miroir du hardware
5. **HAL :** abstraction des périphériques hardware
6. **Common :** paquet virtuel du système



3.4 Factory

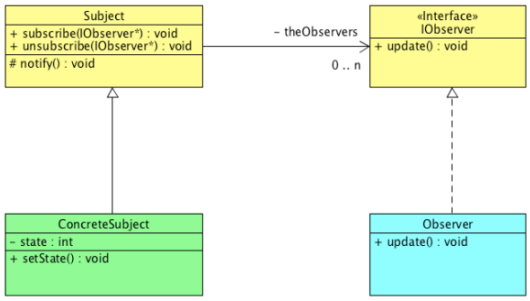


Dans `buildSystem()` :

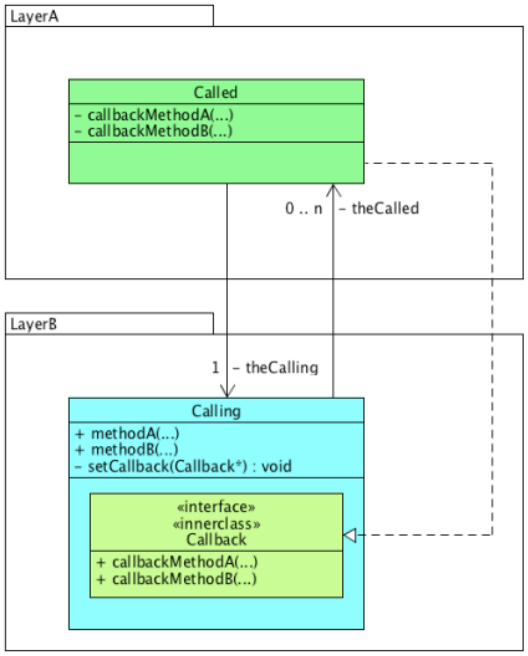
1. `create`
2. `initialize`
3. `initializeRelations` quand tous les `create` et `initialize` sont faits

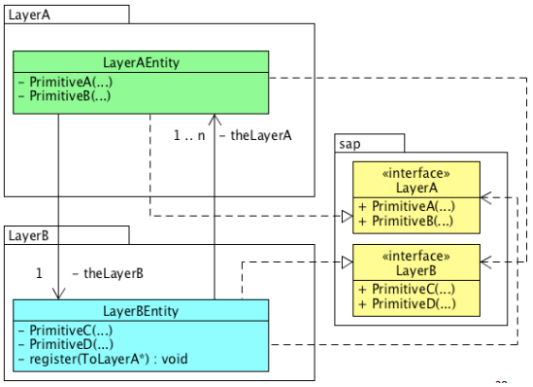
3.5 Observateur

Dérivé du pattern *Observer*.



3.6 Interface SAP (Service Access Point)





4 XF et Machines d'états

Une machine d'état est composée d'un nombre fini :

1. d'états, qui peuvent avoir des actions :
   (a) sur l'entrée
   (b) continue
   (c) sur la sortie
2. de transistions :
   (a) déclenchée par un événement,
   (b) peuvent avoir condition,
   (c) et peuvent exécuter une action

Ces éléments sont signalés via des décorateurs : `trigger[guard]/action`
Exemple :

4.1 XF

Le XF est une couche d'abstraction pour l'OS (interface seulement) ou offre des services similaires à un OS (interface + implémentation dans ce cas).
Opération atomique : opération garantie d'être exécutée sans interruptions (→ une seule étape).

4.2 ISI double switch SM pattern

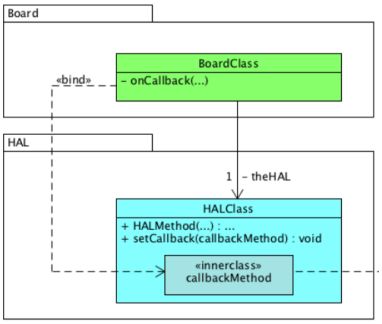
Machine d'état à double switch :

— Un switch de transition
— Un switch d'action

5 HAL

Couche d'abstraction hardware.
CMSIS - Cortex Microcontroller Software Interface Standard : standardisation sur la façon dont les développeurs voient un microcontrôleur et ses périphériques.
Assure :

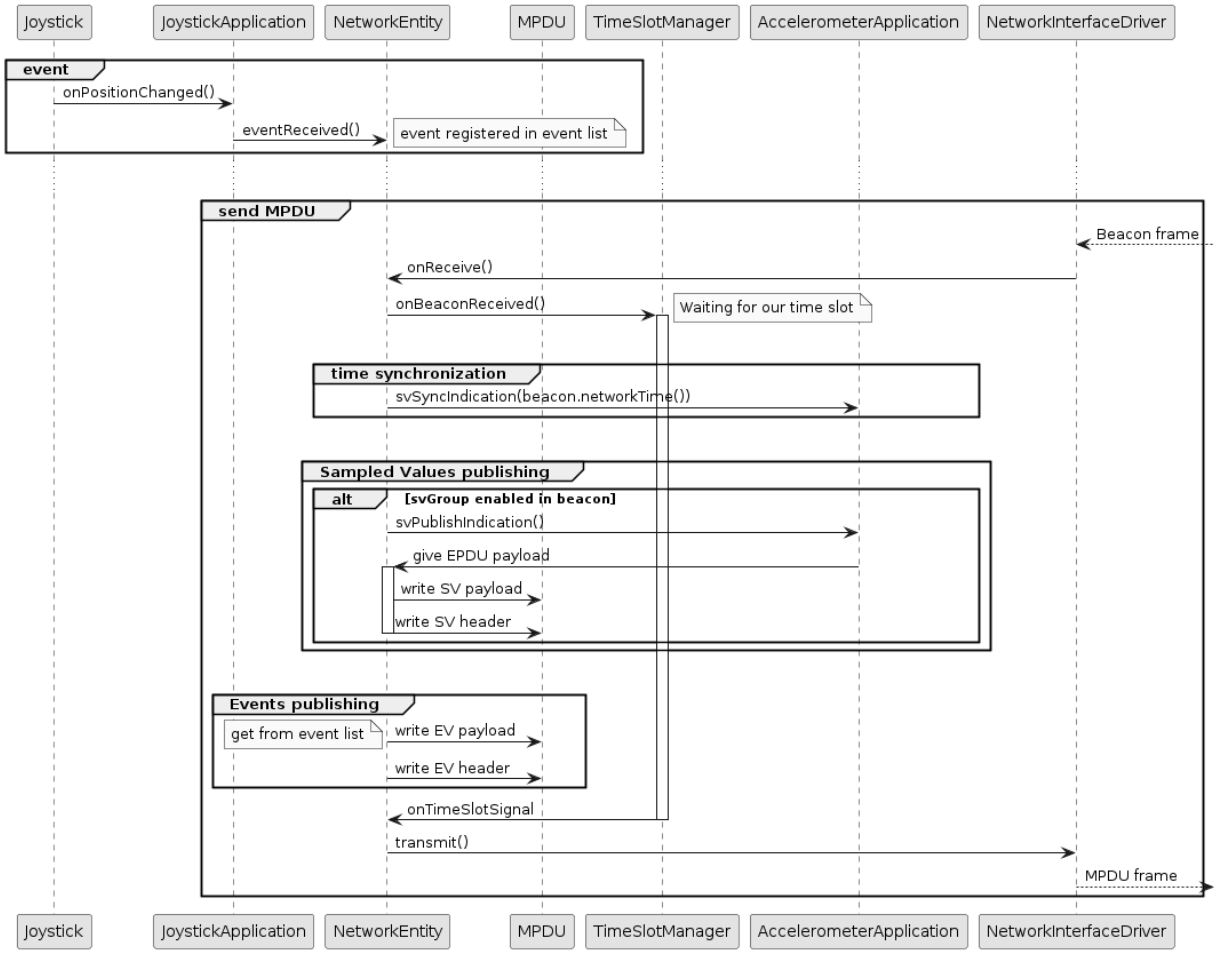
1. Portabilité
2. Réutilisable



5.1 Couches OSI

N	Nom	unité	description	exemples
7	Application (*Application*)	Donnée	Utilité pour l'utilisateur (transfert de fichiers, vidéos, etc...)	FTP, IMAP, HTTP
6	Présentation (*Presentation*)	Donnée	Formats, mises en formes, cryptage, login	JSON, ASCII, HTML, Unicode
5	Session (*Session*)	Donnée	Gestion de l'activité	RPC, NetBios
4	Transport (*Transport*)	Segment, Datagramme	sous-adressage, communication entre deux processus	notion de port (TCP, UDP)
3	Réseau (*Network*)	Paquet	Addressage logique (IP addr.)	IPv4/IPv6
2	Liaison (*Data Link*)	Trame	Adressage physique (MAC addr.)	Ethernet, CAN
1	Physique (*Physical*)	Bit, Symbol	Signaux électriques	Filaire ou sans-fil

Les couches 1-3 sont les couches matérielles. Les couches 4-7 sont les couches hautes liées à l'utilisation faite des données.


Joystick
JoystickApplication
NetworkEntity
MPDU
TimeSlotManager
AccelerometerApplication
NetworkInterfaceDriver
event
onPositionChanged()
eventReceived()
event registered in event list
send MPDU
Beacon frame
onReceive()
onBeaconReceived()
Waiting for our time slot
time synchronization
svSyncIndication(beacon.networkTime())
Sampled Values publishing
alt
[svGroup enabled in beacon]
svPublishIndication()
give EPDU payload
write SV payload
write SV header
Events publishing
get from event list
write EV payload
write EV header
onTimeSlotSignal
transmit()
MPDU frame
Joystick
JoystickApplication
NetworkEntity
MPDU
TimeSlotManager
AccelerometerApplication
NetworkInterfaceDriver