Les principaux domaines de l'informatique

... abordés dans le cadre de ce cours:

- La Programmation
- Les Systèmes d'Exploitation
- Les Systèmes d'Information
- La Conception d'Interfaces
- **Les Agents Logiciels**
- Le Calcul Scientifique

Pourquoi des agents logiciels ? (1)

Du fait de la complexification croissante des tâches à informatiser, les interactions avec les systèmes informatiques deviennent de plus en plus complexes.

Elles se caractérisent en particulier par une **richesse fonctionnelle** et des **possibilités de paramétrage** de plus en plus importantes et il devient de ce fait **difficile de concevoir des interfaces figées**.

La nécessité **d'interfaces facilement programmables** devient de plus en plus incontournable, et, avec l'avènement et la généralisation de l'approche objet et du **principe d'encapsulation** qui lui est associé, cette «programmabilité» peut désormais être conçue comme une interaction avec des entités logicielles (les objets), par le biais des **interfaces de communication**/programmation (API – *application programming interfaces*) qu'elles proposent.

Pourquoi des agents logiciels ? (2)

Cependant, les utilisateurs, qui ne sont pas nécessairement des programmeurs, manquent souvent de formation et l'utilisation des systèmes informatiques directement par le biais des API n'est pas réaliste.

On est donc confronté à un problème:

⇨ d'un côté la nécessité de concevoir des interfaces suffisamment flexibles et donc programmables;

⇨ de l'autre, l'incapacité de la majorité des utilisateurs à utiliser effectivement les interfaces de programmation fournies, qui restent trop complexes.

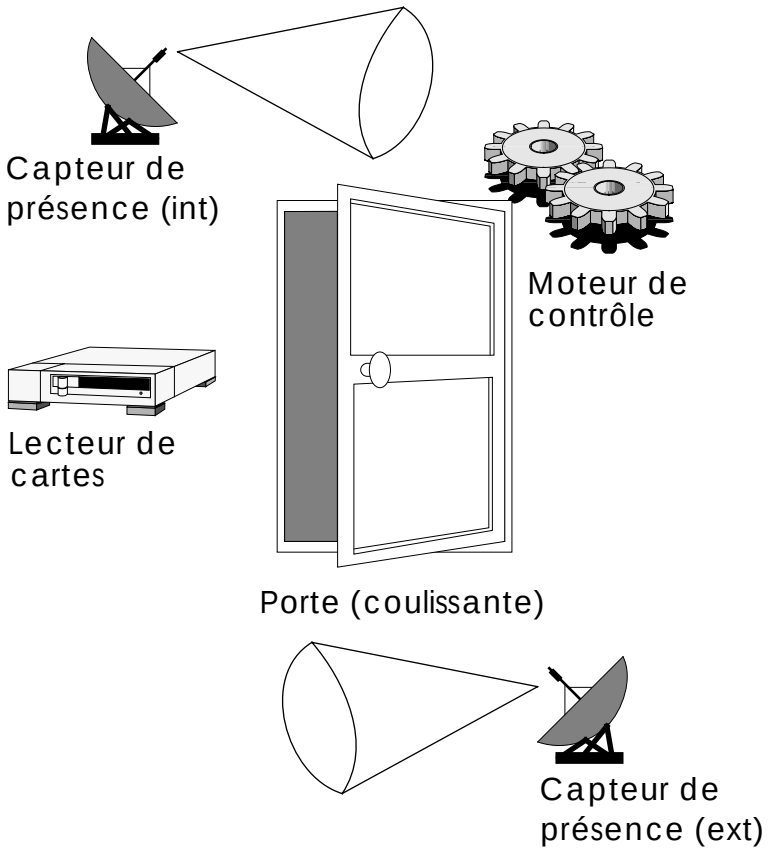
Il faut donc modifier l'approche traditionnelle de l'interaction personne-machine en y **intégrant plus d'intelligence et d'automation**

c'est l'un des rôles importants des ***agents logiciels***.

Exemple: contrôle de porte (1)

Un exemple simple: le mécanisme de contrôle d'une porte automatique

Considérons un système relativement simple de contrôle de porte automatique constitué des éléments suivants :

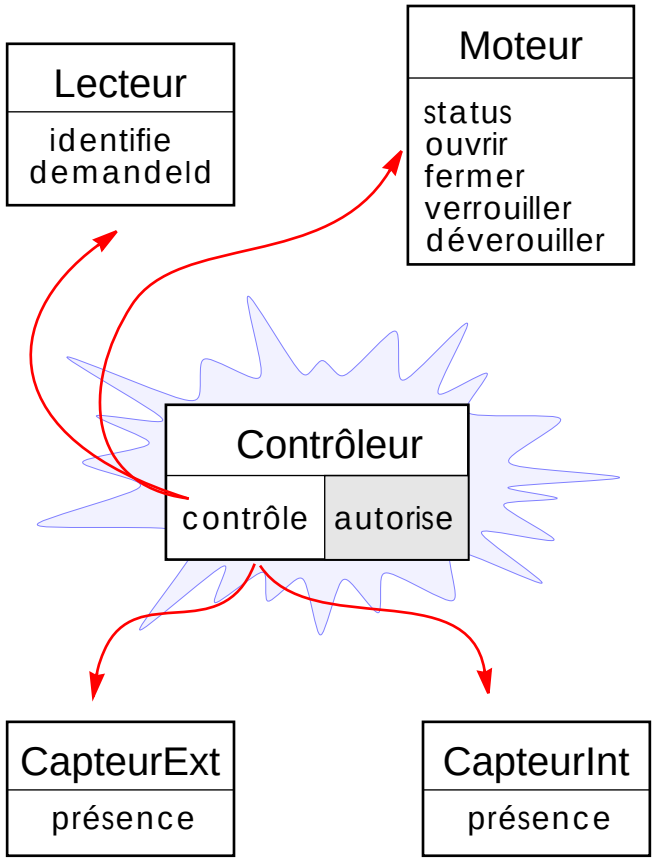


- le **moteur de contrôle** de la porte coulissante: ce moteur doit permettre quatre actions principales: l'ouverture, la fermeture, le verrouillage de la porte (blocage en position fermée, si la porte n'est pas verrouillée, alors elle doit s'ouvrir dès qu'une présence est détectée) et son déverrouillage;
- deux **capteurs de présence** (un de chaque côté de la porte) permettant de détecter une présence dans une zone active déterminée autour de la porte;
- un **système d'authentification** (par exemple un lecteur de carte) permettant à un éventuel utilisateur de valider son identité pour déterminer ses droits d'accès.

Exemple: contrôle de porte (2)

Ce système physique peut être modélisé sous la forme de 5 objets logiciels:

- un objet **`porte`** disposant de 5 méthodes: `ouvrir()`, `fermer()`, `verrouiller()`, `déverrouiller()` et `status()`;
- deux objets **`capteurs`** [`capteurInt` et `capteurExt`] disposant chacun d'une méthode `présence()` (qui renvoie une valeur *vraie* si une présence est détectée, *fausse* sinon)
- d'un objet **`lecteur`** disposant de 2 méthodes: `demandeIdentification()` dont le rôle est de signaler à l'utilisateur qu'il doit s'identifier (p.ex. en faisant clignoter un signal lumineux sur le lecteur) et `identifie()` qui réalise l'identification et renvoie un identificateur d'utilisateur en cas d'identification réussie ou une valeur nulle sinon.
- d'un objet **`contrôleur`**, dédié au pilotage de la porte, disposant de la méthode centrale `contrôle()`, ainsi que de la méthode `autorise()`, indiquant si la personne identifiée a ou non le droit d'entrer.



Exemple: contrôle de porte (3)

A l'aide de ces objets, on pourrait par exemple écrire le programme de contrôle très simple suivant:[1]

```
void contrôle() {
   while (VRAI) {
      if (capteurInt.presence()==VRAI) {
         porte.ouvrir();
         for (int t=0; (capteurInt.presence()==VRAI) && (t<TIME_OUT); t++);
         porte.fermer();
      }
      if (capteurExt.presence()==VRAI) {
         int userId(0);
         if (porte.status()==VERROUILLEE) {
            lecteur.demandeIdentification();
            userId = lecteur.identifie();
         }
         if ((porte.status()!=VERROUILLEE) || (autorise(userId)==VRAI)) {
            porte.ouvrir();
            for (int t=0; (cptInt.presence()==VRAI) && (t<TIME_OUT); t++);
            porte.fermer();
} } } }
```

boucle infinie permettant un contrôle continu

détection d'une présence pour sortir

on doit toujours pouvoir sortir

détection d'une présence pour entrer

boucle d'attente de sortie (avec temporisation)

si la porte est verrouillée, on demande une identification de l'utilisateur extérieur

boucle d'attente d'entrée (avec temporisation)

1. Notez que ce programme très simple ne gère par le verrouillage et le déverrouillage de la porte. De plus, l'association entre une porte, un lecteur et un couple de capteurs doit être faite par le programmeur, lors du choix des arguments du constructeur du contrôleur.

Limitations de l'approche à base d'objets

L'implémentation très simple sous forme d'objets que nous avons esquissée possède plusieurs limitations qui la rendent difficilement extensible à des scenarii de contrôle plus sophistiqués :

⇨ la **logique** du contrôle est **noyée dans le corps de la méthode** `contrôle`: cela la rend difficile à maintenir et à faire évoluer. Il serait plus efficace de pouvoir exprimer cette logique de façon déclarative, indépendante de sa mise en œuvre au sein d'un programme;

⇨ le déroulement du contrôle n'est pas très naturel, ce qui ne facilite pas sa mise en œuvre par des utilisateurs non avertis.

Pour illustrer le second point, considérons ce qui pourrait se passer si le contrôle était fait (comme au bon vieux temps) par un portier humain (par exemple assis dans une loge de verre avec deux fenêtres de communication, une vers chacun des deux cotés de la porte.

Un scénario naturel (1)

1. Cas d'un utilisateur sortant:

Le portier voit arriver de l'intérieur un utilisateur vers la porte; lorsque celui-ci est près de lui, il lui dit «*au revoir et bonne journée*» et il déclenche l'ouverture de la porte; l'utilisateur lui répond «*merci, bonne journée à vous aussi*» et passe la porte que le portier peut alors refermer.

De même, si le portier voit arriver un utilisateur traînant un gros carton plein de livres, le processus sera à peu près similaire, sauf que le portier attendra un peu plus longtemps pour refermer la porte laissant plus de temps à l'utilisateur chargé pour la franchir.

Par contre, si le portier voit arriver un utilisateur avec un café qui se dirige d'un pas lent vers la porte, il lui demandera probablement «*aimeriez-vous sortir ?*» et si la réponse est «*non, je regarde seulement quel temps il fait dehors*» alors le portier laissera la porte fermée même si l'utilisateur est tout près cette dernière (zone dans laquelle le portier aurait pourtant normalement ouvert la porte pour laisser sortir un utilisateur «normal»).

Un scénario naturel (2)

2) cas d'un utilisateur entrant

Lorsque le portier verra arriver de l'extérieur un utilisateur vers la porte, il lui demandera probablement «*aimeriez-vous entrer ?*». Si la réponse est affirmative, alors le portier pourra décider (par exemple selon l'heure) de demander ou non une identification. Par contre, si l'utilisateur répond quelque chose comme «*est-ce bien le bâtiment informatique ?*» alors le portier lui donnera une réponse plutôt qu'une demande d'identification !...

Si l'identification est nécessaire, le portier pourra par exemple demander un pièce d'identité pour consulter une liste de personnes autorisées et, si l'accès n'est pas autorisé (ou si l'utilisateur n'a pas de pièce d'identité), alors le portier devrait pouvoir conseiller un démarche à suivre pour débloquer la situation.

Analyse des scenarii naturels (1)

Si, dans les scenarii présentés, on modélise le portier et l'utilisateur respectivement par deux entités logicielles, on voit facilement que le déroulement du contrôle sera assez substantiellement différent de celui esquissé dans la méthode contrôle.

Parmi les différences importantes, on peut remarquer:

- Une **interaction constante** entre les deux entités, qui ne se contentent pas d'invoquer des commandes (méthodes disponibles) mais participent à un véritable dialogue; cet état de fait permet de gérer des situations complexes, en particulier des situations où se produisent des cas de figure non prévus (par exemple le badaud qui vient boire son café devant la porte sans vouloir sortir).

- Un **comportement pro-actif** de la part des entités (en particulier pour ce qui est du dialogue): l'objet portier prend un certain nombre d'initiatives et ne fait pas que réagir de façon passive à son environnement (par exemple lorsqu'il referme la porte plus lentement pour un utilisateur chargé que pour un utilisateur normal);

- Le **comportement** des entités est fortement **inter-dépendant** avec un environnement précis (utilisation d'appels à des méthodes liés à des éléments très spécifiques de l'environnement – lecteur, porte, ...).

Analyse des scenarii naturels (2)

On pourrait bien sûr chercher à sophistiquer l'approche à base d'objets purs,

de façon à intégrer les différents éléments permettant de rendre le contrôle à la fois plus naturel (en y intégrant les notions de but et de comportement par exemple) et plus performant.

Cela se traduirait cependant par une complexité d'implémentation très importante...

... il faut donc mieux plutôt envisager la mise en œuvre d'une technique plus adaptée : les **agents logiciels**.

Qu'est-ce qu'un agent logiciel ?

Les agents logiciels ont été introduits en Intelligence Artificielle.

Ils permettent l'implémentation concrète
de plusieurs notions développées en IA, en particulier
les notions de **raisonnement automatique** (systèmes experts,
raisonnement à base de cas, ...), de **planification** (détermination des actions
à réaliser pour atteindre un but), **d'analyse du discours** (actes de langage), ...

Ils sont souvent implémentés sous la forme
d'**entités anthropomorphiques**, i.e. des entités artificielles dont
le comportement est inspiré de l'analyse de comportements humains
(notions de croyances, désirs et intentions - BDI)

Le plus souvent, **les agent logiciels sont implémentés à l'aide d'objets**.

Ils peuvent donc être vus comme une des extensions importantes de cette approche,
et l'on peut dire que l'approche agent correspond à un niveau d'abstraction
plus élevé que l'approche object

Agents logiciels: 1ère caractérisation (1)

Un agent logiciel est un système informatique:

- **intégré** (dans un environnement applicatif particulier) : ce sont des systèmes qui doivent intégrer et prendre en compte dans leurs représentation interne des informations précises sur leur environnement;

- **autonome** : il peut agir sans l'intervention des autres – humains ou autres agents – et a le contrôle sur ses actions et ses états internes;

- **réactif** et **pro-actif** : il doit pouvoir percevoir son environnement afin de réagir à des modifications de ce dernier (réactivité), et être capable d'entreprendre des actions en fonction de ses buts, pas seulement en réponse à des stimuli extérieurs (pro-activité);

- **communicant** : il doit être capable d'interagir, lorsque c'est nécessaire, avec des humains ou avec d'autres agents.

Agents logiciels: 1ère caractérisation (2)

Concrêtement, un agent logiciel est un système informatique :

- à **longue durée de vie** : ce n'est pas juste un programme qu'on lance uniquement lorsque c'est nécessaire;
- qui possède des senseurs (**capteurs**) et des moyens de réaliser des actions sur son environnement (**effecteurs**);
- qui est muni d'un **protocole de communication** avec son environnement (humains ou autres agents);
- qui possède une **représentation interne de son environnement**;
- qui admet des **buts**: il doit donc intégrer une certaine forme d'intelligence lui permettant de **décider de façon autonome** des actions à entreprendre pour atteindre ces buts; il doit donc être capable de raisonner et de planifier ses actions en fonction de ses objectifs et de l'état courant de son environnement.

Les agents logiciels: intégration

L'intégration d'un agent logiciel dans son environnement (applicatif) est une notion importante.

Elle signifie en particulier que le comportement de l'agent va être spécifique à cet environnement et que c'est de cette adéquation particulière qu'il va tirer une grande partie de son efficacité.

Concrètement, l'intégration d'un agent se traduit par le fait :

- ☞ qu'il doit disposer d'une représentation interne explicite de son environnement ;
- ☞ qu'il doit disposer d'une identité propre qui le rend unique dans l'environnement considéré ;
- ☞ qu'il doit pouvoir effectivement interagir avec son environnement (présence de senseurs et d'effecteurs).

Dans le cas des agents logiciels (*software agents*), l'environnement est constitué des entités informatiques (programmes, autres agents, serveurs, ...) accessibles à l'agent, par exemple par le biais d'un réseau

Les agents logiciels: autonomie

Que cela veut-il dire d'être autonome pour une entité logicielle qui est de toutes les façons déterminée par le code informatique qui la constitue ?

Dans le cas des agents logiciels, cela signifie que la **mise en correspondance entre** les **entrées** de l'agent (produites par ses senseurs) et ses **sorties** (actions réalisées par ses effecteurs) n'est **pas immédiate** (i.e. comparable à la simple réponse à une commande):

elle va en fait dépendre, de façon éventuellement complexe, **des buts** de l'agent et sera produite à l'aide de ses **capacités de raisonnement** et **de planification**. L'autonomie se traduit souvent par des capacités d'initiative (pro-activité)

⇨ Le fait que la dépendance entre les entrées et les sorties d'un agent n'est pas mécanique a, en particulier, pour conséquence qu'**un agent peut refuse**r **d'effectuer une action**, si elle n'est pas compatible avec ses buts.

D'une façon générale, le comportement d'un agent est défini en termes de «Comment décider quoi faire» plutôt qu'en termes de «Quoi faire».

Les agents logiciels: communication (1)

La capacité de communiquer est une des caractéristiques essentielles des agents logiciels. Cette capacité est implémentée en donnant à ces agents la possibilité d'**échanger des messages**.

Cependant, pour qu'un échange de messages puisse se traduire en une véritable communication, un langage de communication doit être défini pour les agents (*agent communication language*). Pour cela, plusieurs conditions doivent être vérifiées.

En particulier, doivent être définis:

☞ un **vocabulaire commun** permettant aux agents de faire référence de façon unifiée à un ensemble de concepts et d'actions compris de tous.

Un tel vocabulaire commun est généralement structuré, et s'appelle une ***ontologie*** (partagée); l'existence d'une *ontologie* partagée permet aux agents de pouvoir **affecter un même sens** aux unités élémentaires du langage de communication, les *tokens* (mots).

D'une façon générale, les *ontologies* permettent de définir des domaines d'application en termes d'objets, d'attributs et de relations sous la forme de modèles conceptuels similaires à ceux utilisés dans le domaine des systèmes d'information.

Les agents logiciels: communication (2)

- Un mécanisme permettant de **référencer de façon non ambiguë les objets de l'environnement** (c'est une des raisons pour laquelle chaque agent doit être muni d'une identité propre): *names mapping*.

- Une théorie permettant **d'associer un sens aux messages** échangés entre les agents: une telle théorie devra permettre d'associer des buts (objectifs) concrêts aux actions exprimées dans les messages, ainsi que de déterminer l'impact de ces actions sur les états internes des agents (par exemple par des pré et post-conditions);
  le langage permettant de décrire les états internes des agents pour pouvoir exprimer le contenu des messages échangés est appelé langage de communication de contenu (*content communication langage* ou CCL)

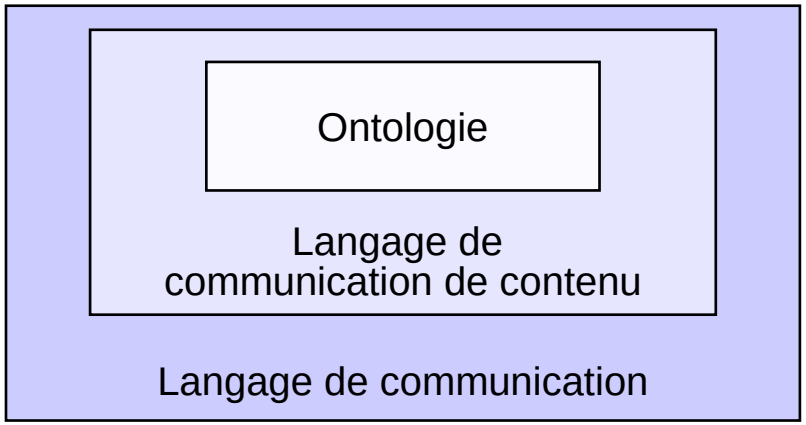
- Un **langage permettant d'exprimer les messages** échangés dans une forme structurée reconnue par l'ensemble des agents de l'environnement; un tel langage est appelé langage de communication entre agents (*agent communication language* ou ACL) ;

- Un **protocole permettant d'organiser les séquences de messages** échangés en de véritables conversations structurées d'une façon similaire à ce que l'on peut observer chez les humains (théorie des actes de langage : requêtes, suggestions, promesses, menaces, ...)

Les agents logiciels: communication (3)

On trouve en particulier des standards pour:

- ☞ les langages de communication de contenu (KIF, FIPA-SL, FIPA-CCL, ...)
- ☞ et pour les langage de communication entre agents (KQML, FIPA-ACL, ...).



Des efforts importants sont également mis en œuvre pour la création d'ontologies et de protocoles liées à des domaines spécifiques (tourismes, commerce électronique, ...).

Un travail de standardisation important est par exemple en cours dans le cadre de la FIPA (*Foundation for Intelligent Physical Agents*, voir `http://www.fipa.org`).

Exemple de type de messages (*Communication Acts*) définis dans FIPA-ACL:

`agree`, `cancel`, `confirm`, `disconfirm`, `failure`, `inform`, `not-understood`, `query-ref`, `refuse`, `request`, `subscribe`, ...

Exemple: message FIPA-ACL

Exemple de message au standard FIPA-ACL
avec un contenu exprimé en FIPA-SL

L'agent *i* confirme à l'agent *j* que la baleine est un mammifère :

```
(confirm
:sender (agent-identifier :name i)
:receiver (set (agent-identifier :name j))
:language FIPA-SL
:content (is mammal whale)
)
```

Autre exemple de message

```
(request
:sender ( :name martin_agent@liawww.epfl.ch:8080)
:receiver (:name movenpick_hotel@tcp://movenpick.com:6600)
:ontology personal_travel_assistant
:language FIPA-SL
:protocol fipa-request
:content
   (action movenpick_hotel@tcp://movenpick.com:6600
   (book-hotel :arrival 25/11/2000 :departure 05/12/2000 ...
)  )
```

Exemple d'interprétation d'action en KQML

[sémantique d'interprétation]

Tell(A,B,Y)

precondition(A):
believe(A,Y) and
know(A,want(B, know(B, (believe(A,Y) or not(believe(A,Y))))))

precondition(B):
intend(B, know(B,(believe(A,Y) or not(believe(A,Y)))))

postcondition(A):
know(A, know(B, believe(A,Y)))

postcondition(B):
know(B, believe(A,Y))

Quelques exemples d'agents logiciels

- les robots autonomes (par exemple kephera);

- certains programmes spécifiques comme le programme Elisa;

- les assistants experts (ou *wizards*) présents dans certaines suites logicielles comme Office.

Différents types d'agents logiciels (1)

Les agents logiciels peuvent être caractérisés par :

- la nature de la tâche qu'ils réalisent:
  - les agents personnels des utilisateurs (leur rôle est d'assiter un utilisateur dont ils connaissent les intérets, les préférences et les habitudes): par exemple les *personal news editors*, les *personal e-shoppers*, ...
  - les agents de service (qui réalisent des tâches plus générales, souvent en tâche de fond) : par exemple les indexeurs du Web (*spiders*, *crawlers*) les gestionnaires de réseau, ...

Différents types d'agents logiciels (2)

☞ la nature de leur intelligence:

- ⇨ **programmée par l'utilisateur** (qui détermine directement les règles de comportement). Généralement simples, souvent pas très intelligents, dépendant des compétences de l'utilisateur lui-même, disponibles commercialement (un exemple très simple: l'outil `procmail`);
  ces agents admettent le plus souvent un comportement stimulus-réponse pur; ils sont alors appelés des **agents réactifs**.

- ⇨ **utilisant des techniques d'intelligence artificielle** (raisonnement à base de cas, système expert, satisfaction de contraintes, ...) souvent très sophistiqués, programmés par des ingénieurs de la connaissance (*knowledge engineers*); pour l'instant pas disponibles à une grande échelle; ces agents implémentant des mécanismes plus complexes d'association entre les entrées (stimulis) et les sorties (réponses) sont appelés **agents délibératifs**.

- ⇨ **intégrant des techniques d'apprentissage** (l'agent détecte des *patterns* dans les actions de l'utilisateur et les met à profit) de complexité moyenne; commencent à devenir disponible (*mail agents*).

- ⇨ **collaborative** (l'intelligence provient de l'interaction entre plusieurs agents).

Exemple de mise en œuvre (1)

[le cas de l'accès à un bâtiment]

Nous allons maintenant illustrer l'approche à base d'agents
à l'aide d'un exemple simple d'accès à un bâtiment.

Définition de l'environnement applicatif :

- la tâche à réaliser est d'assurer un accès (aussi naturel que possible) à la réception d'un bâtiment donné;
- l'environnement (logiciel) est constitué des 4 objets déjà définis (porte, capteur intérieur, capteur extérieur et lecteur).

On pourrait définir trois agents logiciels:

- l'un **responsable du contrôle de la porte** du bâtiment qui nous intéresse (et qui va donc remplacer l'objet `contrôleur` précédemment défini);
- le second servant, pour un utilisateur donné, **d'interface de communication avec l'environnement** applicatif (le rôle de cet agent personnel est donc de décharger autant que possible l'utilisateur des interactions nécessaires pour permettre sa bonne circulation dans l'environnement);
- et le troisième étant un agent de service, **gérant le répertoire des agents disponibles** dans l'environnement considéré.

Exemple de mise en œuvre (2)

[déroulement de l'accès]

Dans le cadre d'une approche à base d'agent, le déroulement de l'accès d'un utilisateur à un bâtiment donné (disons par exemple le bâtiment informatique) pourrait être le suivant:

1. l'utilisateur donne à son agent personnel l'objectif d'atteindre la réception du bâtiment informatique (et c'est la seule information que l'utilisateur devrait avoir à donner à son agent personnel);

2. l'agent personnel contacte l'agent gérant le répertoire des agents recensés dans l'environnement pour lui demander la localisation de la réception du bâtiment informatique;

3. la localisation obtenue permet à l'agent personnel de calculer un itinéraire de déplacement (attention, cet itinéraire est un itinéraire idéal qui ne doit pas avoir à prévoir les éventuels obstacles comme les portes fermées);

Exemple de mise en œuvre (3)

[déroulement de l'accès – suite]

4. la connaissance de l'itinéraire permet à l'agent personnel de déclencher un processus de déplacement (voiturette automatique par exemple) vers la destinatiuon cible;

5. lorsque l'utilisateur arrive dans la zone de détection du capteur extérieur de la porte du bâtiment informatique, l'agent de contôle de la porte est activé; il contacte l'agent personnel de l'utilisateur (envoi d'un message) pour lui signaler un obstacle (porte fermée); l'indication de la présence d'un obstacle entraîne, de la part de l'agent personnel, une interruption momentanée du processus de déplacement et une demande de suppression de l'obstacle (notez que l'agent personnel ne doit pas forcément savoir comment éliminer un obstacle du type porte fermée; la solution à ce problème – demande d'ouverture de la porte – peut tout à fait être connue du seul agent de contrôle de la porte)

6. l'agent de contrôle de la porte accuse réception de la demande de suppression d'obstacle, la traduit en une demande d'ouverture de la porte ce qui entraîne le déclenchement par l'agent de contrôle d'une procedure d'authentification d'identité; il envoie une demande de code d'identification à l'agent personnel puis transmet le code reçu à l'objet lecteur qui le valide;

Exemple de mise en œuvre (4)

[déroulement de l'accès – suite]

7. l'agent de contrôle de la porte déclenche l'ouverture de la porte et signale à l'agent personnel la disparition de l'obstacle; l'agent personnel accuse réception de ce message et ré-enclenche le processus de déplacement;

8. lorsque l'utilisateur arrive dans la zone de détection du capteur intérieur, l'agent de contrôle est de nouveau activé et déclenche la fermeture de la porte;

9. lorsque l'utilisateur atteint la réception (i.e. la localisation recherchée), son agent personnel lui indique que l'objectif a été atteint et se met en attente d'une nouvelle demande.

Exemple de mise en œuvre: conclusion

Plusieurs aspects caractéristiques d'une approche à base d'agents sont à noter dans le scénario précédent:

- ☞ sa relative **complexité** (l'implémenter de façon complète sous la forme d'une ou de plusieurs methodes sera passablement délicat);
- ☞ la relative **simplicité des comportement individuels** des agents (dont les capacités peuvent être limitées aux tâches pour lesquelles ils sont effectivement responsables)
- ☞ le remplacement d'une solution globale prédéterminée et calculée *a priori* par une **séquence de solutions partielles calculées itérativement** de façon collaborative par les agents concernés (dont aucun, pris isolément, n'a la compétence suffisante pour produire la solution globale);
- ☞ l'importance de la **communication entre les agents** qui apparaît comme un ingrédient incontournable pour le bon fonctionnement de l'approche à base d'agents;

Ces aspects montrent tout l'intêret d'une approche à base d'agents dont la mise en œuvre effective est aujourd'hui facilité par l'émergence de plateformes de développement (comme par exemple la plateforme JADE, ou JatLite) spécifiquement dédiées au développement d'environnements d'agents.