Université de Sherbrooke

Faculté des sciences

Centre de formation en technologies de l'information

CeFTI

INF756

Accompagnement au plan de cours

Note : ce document accompagne le plan de cours et vise à apporter de l'information contextuelle sur le cours. L'information d'ordre administratif est disponible dans le plan de cours en soi.

Avant d'entreprendre ce cours…

Le cours présume chez l'étudiant(e) une connaissance de la programmation, et mettra l'accent sur les parallèles entre le modèle OO et le modèle CS, qui se rejoignent à plusieurs égards, en particulier dans le cas des approches par composants.

Puisque l'aspect « serveur » sera entre autres illustré à l'aide d'accès à une BD, le cours présume aussi une familiarité de l'étudiant(e) avec les SGBD et le langage SQL, ou du moins qu'elle ou il le deviendra en chemin.

Enfin, le cours présume chez l'étudiant(e) des aptitudes analytiques se prêtant à la spécialisation et à l'optimisation de processus. Nous ne cheminerons pas beaucoup en ce sens, faute de temps, mais nous serons préoccupés par la mise au point de processus efficaces, et par l'impact de l'efficacité de certains modules sur celle des autres.

Il est hautement probable que ces différents *a priori* soient acquis par chacune et chacun d'entre vous compte-tenu des préalables d'entrée au DGL. Personne n'est parfait, cela dit, alors envisagez fortement vous adjoindre des coéquipières et des coéquipiers ayant des forces complémentaires aux vôtres.

Oui, mais j'ai déjà vu cela…

Il y aura inévitablement des éléments de votre formation au DGL (ou à la MTI), dans ce cours comme ailleurs, qui recouperont votre vécu académique ou professionnel. C'est sans doute inévitable dans les circonstances : il est présumé que vous avez au préalable une formation informatique universitaire de même qu'un peu d'expérience sur le terrain.

Prenez note que la formation préalable des individus de votre groupe est variée, et que ce qui paraît évident pour l'un(e) d'entre vous peut paraître très ardu ou très étrange pour l'autre. Selon les cours, probablement même selon les éléments d'un même cours, les perceptions individuelles risquent fort de varier alors faites attention aux jugements trop rapides. Vous n'avez qu'à penser à quelqu'un qui a beaucoup œuvré dans le monde des BD en comparaison avec quelqu'un dont l'expérience se rapproche plus des systèmes embarqués, pour prendre un cas vécu.

Un diplôme comme le DGL cherche à utiliser la variété des acquis et des horizons comme levier pour rendre la formation encore plus pertinente. C'est un privilège, profitez-en!

Mise en contexte

Le cours INF756 existe sous une forme ou l'autre depuis les tout débuts des programmes technologiques offerts au CeFTI[1]. À travers sa formule particulière, par la complexité des connaissances préalables attendues des étudiant(e)s et par la richesse du sujet, ce cours a la particularité d'avoir à la fois instruit et (il faut bien le dire) fait travailler très fort (c'est un euphémisme) un grand nombre de cohortes déjà.

Depuis sa conception et ses premières incarnations, INF756 a plusieurs fois changé de formule pédagogique et de contenu. Explorant un sujet omniprésent (à l'heure d'Internet), incontournable et changeant[2], se donnant comme mandat d'amener l'étudiant(e) à comprendre à la fois par la théorie et par la pratique ses grands enjeux, INF756 est un cours en perpétuelle mutation.

Ce qui ne change pas, toutefois, c'est la pertinence du sujet. INF756 explore les questions relatives à l'architecture, au développement, à l'entretien, à l'acquisition, au déploiement et à la sécurisation des systèmes répartis. Ceci inclut les architectures orientées services (*Service-Oriented Architectures*, SOA), les systèmes client/ serveur (SCS) selon les dénominations classiques et selon plusieurs points de vue controversés, les systèmes point à point, les services Web, l'intégration de systèmes existants et hétérogènes, le style architectural REST, les microservices, *etc*. Ce cours explore concrètement la question de l'informatique contemporaine, et ce sous plusieurs lorgnettes.

La question du nom

Le nom SCS lui-même est controversé. Pour certain(e)s, il s'agit d'un terme obsolète pour une approche désuète; pour d'autres, il s'agit d'un terme au sens pointu et limité pour lequel il n'y a pas lieu de consacrer un cours de 2e cycle universitaire; pour d'autres encore, Internet a tué l'approche CS, tout simplement.

Depuis sa création, INF756 vise la compréhension appliquée de l'ensemble des réalités des systèmes répartis à grande échelle. Là où d'autres cours du programme couvrent ce sujet du point de vue de la gestion de projets, ou d'un point de vue analytique de haut niveau, INF756 pose son regard plus près de celles et ceux qui développement de tels systèmes. Le nom SCS sert de levier pour le cours, qui doit (après tout) porter un nom, mais s'intéresse à l'approche derrière ces systèmes, beaucoup plus qu'à des technologies ponctuelles (qui, de toute manière, changent continuellement). Plutôt que de changer sans cesse le nom du cours, le choix a été fait d'adopter une optique large et un mandat inclusif et critique.

[1] En fait, ce cours était offert avant même que le CeFTI n'existe sous ce nom.

[2] Même le nom du cours devrait changer presque annuellement pour rester à la page, ce qui n'est pas peu dire!

L'approche derrière les SCS

Les environnements de réseaux, dont Internet est un exemple probant, sont une réalité contemporaine incontournable. Il en va de même pour la complexité sans cesse croissante des systèmes informatiques que nous sommes susceptibles d'y rencontrer.

Penser de tels systèmes dans leur réalité, dynamique et hétérogène, demande une approche particulière et, à certains égards, déstabilisante. C'est cette approche que nous nommerons l'approche CS, et qui regroupe des savoirs et des savoir-faire incontournables pour qui veut faire sa place en tant qu'architecte système.

Cette approche permet, entre autres, de réfléchir à des questions portant sur :

- La localité des données, qu'il s'agisse d'une centralisation dans un ou des lieux où il sera plus simple d'assurer leur sécurité et leur uniformité, ou d'une distribution pour réduire les probabilités que le système ne dépende que d'un seul de ses nœuds.
- Une séparation des tâches entre plusieurs entités logicielles spécialisées, à des fins de calculs, en vue d'accéder à des masses de données, à des fins de présentation, *etc*. Cette manière de procéder permet une répartition plus efficace des ressources informatiques disponibles.
- Une réutilisation accrue et un meilleur partage des composants logiciels existants, où qu'ils soient, incluant les systèmes hérités (en anglais : *Legacy Systems*).
- Un partage plus efficace entre individus des responsabilités associées au développement d'un système informatique.
- La transformation des systèmes existants pour leur donner une nouvelle vie ou pour augmenter leur rentabilité à moyen terme; *etc*.

Le premier but du DGL[3] est d'amener l'étudiante et l'étudiant à « approfondir ses connaissances sur les méthodes et les outils utilisés pour spécifier, concevoir, implanter et maintenir les systèmes informatiques ». Dans cette optique, comprendre suffisamment les SCS est un avantage concurrentiel important; plus spécifiquement, les comprendre suffisamment pour être en mesure de faire face aux enjeux qui leur sont associés est, pour un(e) architecte technologique, pratiquement vital.

[3] https://www.usherbrooke.ca/admission/programme/550/diplome-de-2e-cycle-en-genie-logiciel/

Objectif général

Le cours INF756 vise à développer chez l'étudiant(e) une ***connaissance appliquée*** des contraintes et particularités spécifiques aux systèmes répartis contemporains, en appliquant une approche CS.

Connaissance appliquée signifiera à la fois le *savoir* permettant de comprendre les systèmes développés selon les principes sous-jacents au modèle client/ serveur, et le *savoir-faire* requis pour en tirer soi-même profit dans un contexte de développement.

En périphérie de cet objectif, nous poursuivrons le développement d'une compréhension opérationnelle d'au moins un langage de programmation (C++, en particulier depuis C++ 11) facilitant la mise en pratique des principes exposés, *sans nous y restreindre*.

Les notes de cours incluent des incursions dans les mondes Java et .NET, entre autres, particulièrement pour les volets Web, et sont axées sur la plateforme *Microsoft Windows* (simple question de marché et de disponibilité des plateformes) sans s'y limiter.

L'approche CS favorise l'emploi de plusieurs langages de programmation distincts dans le développement d'un même système. Nous chercherons à exploiter cette réalité et à mettre l'accent sur les systèmes intégrant plusieurs technologies.

Appliquer plutôt que survoler?

Il arrive que certaines et certains critiquent l'idée de viser une connaissance appliquée dans un diplôme de 2e cycle, privilégiant une approche plus aérienne, plus conceptuelle. Pourtant, le volet appliqué des connaissances acquises ici est la clé de la crédibilité : savoir sans savoir-faire réduirait vos chances d'apparaître crédible aux yeux des gens que vous côtoieriez.

Pour qui ne l'a jamais expérimenté directement, il est difficile d'estimer réellement la différence entre concevoir un système monolithique, traditionnel, et concevoir un système réparti capable de réaliser la même tâche mais en tirant profit du modèle CS. Des considérations apparaissent quant au choix des outils, quant à la sécurité, quant aux considérations d'interopérabilité, quant aux protocoles utilisés, quant à la répartition des tâches entre individus, *etc*. La simple question d'estimer le temps requis pour réaliser une tâche de développement s'aborde de manière complètement différente dans un monde réparti et dans un monde monolithique.

Intellectualiser la différence entre les deux modèles est au moins aussi difficile que ne le serait de comprendre réellement le polymorphisme sans avoir jamais construit une hiérarchie de classes, sans y avoir jamais réfléchi de manière appliquée. Il y a un monde de différence entre pouvoir réciter des différences décrites par autrui et comprendre ces différences au point de pouvoir les apprécier qualitativement ou, mieux, quantitativement.

On peut reconnaître que, malgré les gains clairs qu'apporte l'approche CS dans le processus de développement d'un système complexe, il sera en général plus difficile de concevoir un système réparti que de concevoir un système monolithique à fonctionnalité similaire. Ce qui est plus difficile, c'est de comprendre réellement *pourquoi* (d'où jaillissent ces difficultés, ce qu'elles impliquent) ce sont des difficultés, *quelles sont* ces difficultés et *à quel point* elles se posent.

Il faut entre autres être capable de saisir quand et pour quelle raison appliquer une approche CS (sous une forme ou l'autre) est une décision gagnante et quand cela signifie perdre temps et argent.

Il est probable que, concrètement, vous soyez impliqué(e)s plus aux niveaux conception, acquisition, déploiement et intégration des SCS que vous ne le serez au niveau implémentation, du moins pour la plupart d'entre vous. En retour, il est souhaitable que vous puissiez évaluer le travail requis à d'autres, que vous puissiez faire des choix technologiques et humains éclairés face à des situations impliquant l'achat, le déploiement ou le développement de SCS, et que vous puissiez mesurer des questions relatives à la gestion des ressources (temps, matériel, infrastructures, *etc*.) lorsque confrontés à des choix impliquant l'approche CS. Pour être véritablement capables d'atteindre ces objectifs, il vous faudra vous mouiller un peu.

Mêler les approches

Le modèle CS, du moins sous certaines de ses acceptions, peut être pris comme une extension du modèle OO car, bien que les deux idées soient différentes et n'aient pas *besoin* l'une de l'autre, une tendance répandue est de les marier pour profiter à plein de leurs forces respectives. Les deux approches poursuivent, de toute manière, un ensemble d'objectifs communs[4].

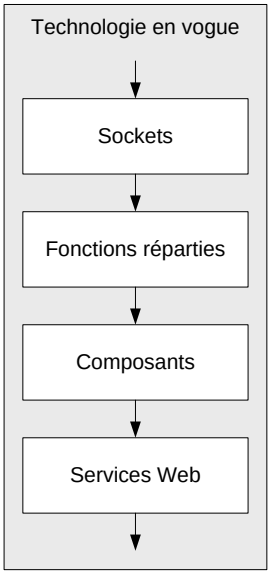
Le modèle CS peut aussi être pris comme une extension du monde des bibliothèques logicielles quand on l'examine sous l'angle des services, Web ou autres. C'est à la fois une tendance ancienne et nouvelle, car (il faut bien l'avouer) le vent tourne souvent et subitement dans le monde des SCS.

Il s'agit d'un modèle qui touche à la fois au logiciel, au matériel et aux humains. Qui demande de poser un regard sur les coûts (financiers ou autres) de croissance et d'évolution des systèmes, sur la sécurité, sur les rôles des intervenants humains et logiciels, sur la complexité et l'entretien des systèmes, sur les protocoles, sur l'évolution du marché… *Plus qu'une simple technique, il s'agit d'une manière différente de penser*, de questionner, de développer, de voir les systèmes, surtout lorsque ces systèmes prennent de l'ampleur et se complexifient.

En ce sens, le design de SCS interpelle à la fois les décideurs et les développeurs. Peut-être plus que tout autre type de design logiciel, celui des SCS s'inscrit dans le temps et pose la question de la situation de l'entreprise et de son infrastructure technologique passée, présente et (bien entendu) future.

[4] Parmi lesquels on trouve : réutilisation du code; coût d'entretien faible; système facilement évolutif; facilité à développer et à gérer des systèmes complexes; robustesse accrue; facilité de débogage; *etc*.

Axes d'étude

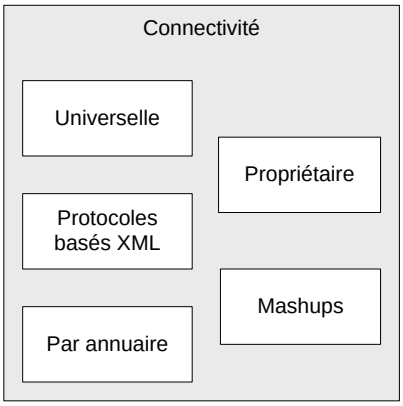


L'approche CS peut être étudiée selon plusieurs axes.

L'un d'entre eux est l'**axe technologique**. Les technologies, les outils influencent notre manière de pensée et constituent des métaphores pour exprimer nos idées. Nous examinerons dans ce cours quelques grandes familles de technologies utilisées pour implémenter des SCS. Des *sockets* bruts aux fonctions réparties, sur lesquelles sont construits les composants, jusqu'aux services Web en vogue aujourd'hui.

Un examen historique et technique de ces grandes familles technologiques nous permettra entre autres de mettre en relief ce qu'elles ont de commun et de semblable, de même que ce en quoi elles diffèrent. Nous développerons un regard critique face au marketing et à ce que certains présentent comme des nouveautés : il y a plus d'éléments qui rapprochent ces diverses technologies qu'il n'y en a qui les éloignent.

Le schéma présentant l'axe technologique ci-dessus est un schéma historique grossier. Plus récent ne signifie pas nécessairement meilleur *pour vous*, mais peut signifier plus en vogue que les approches précédentes.



La manière d'assurer la **connectivité** jette lui aussi une lueur importante sur les mouvances technologiques sous-jacentes aux systèmes répartis. Une fois la base admise, soit le fait qu'au moins un (parfois deux, selon les approches) des homologues d'une relation doit être repérable et identifiable par l'autre, la question du comment de la connectivité demeure ouverte et les options sont nombreuses.

La question n'est évidemment pas de savoir si l'industrie *souhaite* la connectivité[5] mais bien de savoir *comment* la connectivité sera réalisée suite à l'adoption (ou à l'intégration) d'une technologie ou d'une approche

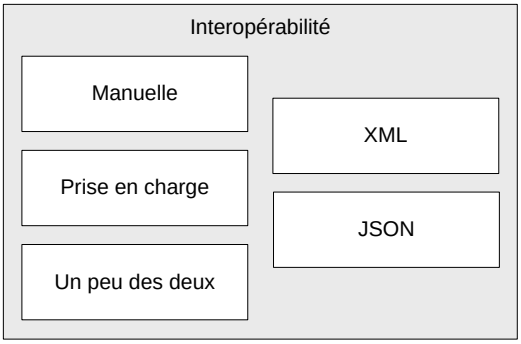
L'offre technologique des intergiciels, très forte et très importante il y a une dizaine d'années à peine, tend à perdre de l'importance face aux approches plus légères (en particulier celles reposant sur les langages dynamiques, tels que JavaScript, et sur des *mashups*) qui font le succès des petites entreprises très agiles qui profitent à fond d'Internet.

Ce qui est « bien vu » change et fluctue au fil des modes et des technologies; une compréhension des enjeux est nécessaire pour développer une vision personnelle de l'évolution technologique.

[5] Cela va de soi, à sécurité équivalente bien entendu (ce qui est moins simple qu'il n'y paraît).

L'**axe d'interopérabilité** est un axe primordial puisqu'il influence la manière qu'on deux entités de transiger entre elles.

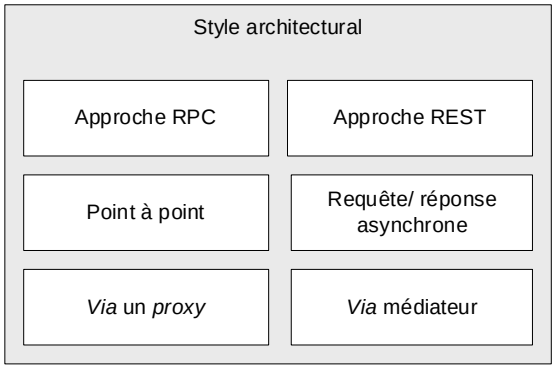
Les stratégies d'interopérabilité varient selon les technologies. Certaines demandent une gestion presque complètement manuelle des questions d'interopérabilité (les *sockets* bruts), d'autres remettent l'interopérabilité entre les mains d'une infrastructure (les composants font typiquement partie de ce groupe) et d'autres encore nécessitent une part de prise en charge et une part d'écriture manuelle (les services Web tendent à tomber dans ce camp, mais les implémentations varient beaucoup ce qui fait qu'il est difficile de les caser clairement dans un camp ou l'autre).



L'interopérabilité est le Graal de l'informatique contemporaine. La majorité des avancées dans le monde des systèmes répartis visent en partie à rendre l'interopérabilité plus simple et plus naturelle, mais la question est beaucoup moins simple qu'il n'y paraît et l'interopérabilité pleine et entière dans un univers hétérogène[6] demande toujours une part d'adaptation et d'effort individuel. Comme le disent les anglophones : *There's No Free Lunch*.

L'informatique contemporaine est un monde complexe et hétérogène. Même si le parc d'une entreprise est homogène, le besoin de communiquer avec celui des clients et des partenaires commerciaux assure la pérennité de l'importance de la connectivité et de l'interopérabilité.

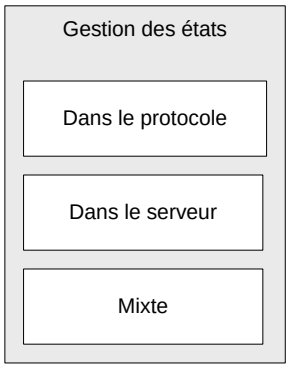
Les **styles architecturaux** possibles pour une approche CS sont nombreux, et incluent des échanges directs, par flux, trames ou invocations de sous-programmes distants, à travers un intermédiaire direct, un médiateur, et ainsi de suite. Ici comme ailleurs, l'approche parfaite n'existe pas, et le *syndrome de la saveur du jour* sévit avec vigueur.



Le danger, avec ces approches, est de penser que l'une est la remplaçante ou l'évolution directe (au sens de « est meilleure que ») de l'autre. La situation est plus subtile et toutes les approches ont leurs avantages et leurs inconvénients. Un changement technologique ici peut entraîner le retour d'une technologie mise de côté pendant quelques temps.

[6] Et non, il n'est pas vrai que de n'utiliser qu'une seule technologie, quelle qu'elle soit, réglerait la question de l'interopérabilité. La gestion des versions et l'évolution technologique demeureraient des réalités. Il faut accepter la nature intrinsèquement complexe des systèmes répartis et aborder les problèmes qui en surgissent avec prudence, intérêt, sérénité et préparation.

La question des **états** et de leur gestion est une question importante : où se situeront les états d'un système? Du seul côté client? Du seul côté serveur? Seront-ils répartis d'un côté comme de l'autre? Ces choix influencent à la fois la sécurité et l'*échelonnabilité* du système.

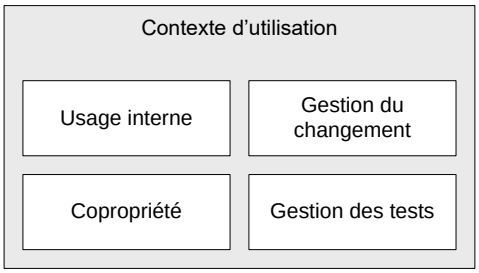


Les protocoles en vogue sur Internet, dont `http`, sont des protocoles sans état, qui forcent les clients à fournir suffisamment d'information au serveur lors de chaque requête pour faire en sorte que ce dernier puisse savoir où en est la transaction en cours. Les choix technologiques peuvent donc être influencés par des besoins de gestion des états, et l'inverse est aussi vrai.

L'approche par services, qui teinte certains des plus importants mouvements technologiques récents dans ce secteur (approche `REST`, services Web, approche `SOA`) présume dans la majorité des cas que chaque requête est complète en elle-même et ne présume aucun état côté serveur. Ceci contraste avec une approche reposant sur des objets répartis, du fait qu'on présume souvent les objets responsables de l'intégrité de leurs états, en vertu du principe d'encapsulation.

On pourrait sans doute trouver plusieurs autres axes à exploiter dans un cours comme celui-ci. Nous sommes sur la ligne de front des changements technologiques les plus rapides.

Le **contexte d'utilisation** d'un `SCS` influencera les choix technologiques et les pratiques associées au système en question. Par exemple, un `SCS` destiné à un usage interne sera peut-être soumis à des contraintes d'interopérabilité mieux définies et moins complexes[7] que son équivalent dont les services seraient exposés à des partenaires commerciaux.



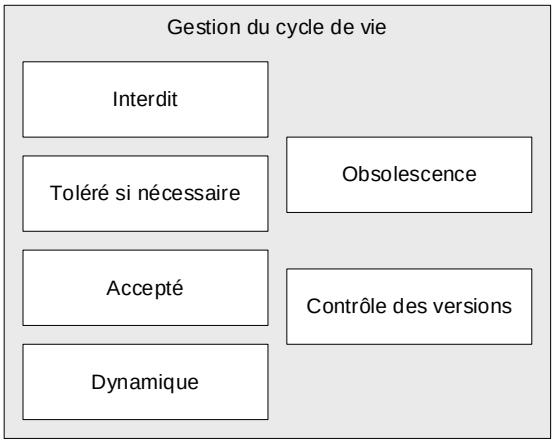
Les systèmes répartis contemporains tendent à être exposés à la fois à l'interne et à l'externe pour une entreprise donnée. Cela introduit des particularités politiques, comme par exemple le partage des coûts et des risques lors de tests, ou encore la gestion collaborative du changement, particularités qu'on aurait eu tendance à négliger par le passé.

[7] … quoique…

La **gestion du cycle de vie**, d'ailleurs, est une question en soi digne d'intérêt. Permet-on la modification des interfaces et des contrats des services exposés par un système? Si oui, quelles sont les conséquences? Qui paiera?

La reconnaissance d'un changement de contrat et l'adaptation à ce changement seront-ils statiques ou dynamiques? Devra-t-on maintenir plusieurs versions du même service? Si oui, combien de versions voudra-t-on maintenir concurremment? Et pour combien de temps?

Que fait-on des services vieillissants? A-t-on planifié l'éventuelle désuétude d'une interface dans le cycle de vie de notre système?



Toutes ces questions, et bien d'autres[8], occuperont nos pensées au cours des prochaines semaines.

Optiques choisies et rejetées pour le cours

Bien comprendre la plupart des modes de pensée demande qu'on les applique suffisamment pour saisir leurs nuances. Il en va ainsi du modèle CS : se limiter à une approche strictement aérienne est un bon moyen de mal paraître.

Par conséquent, ce cours s'accompagnera d'une activité pédagogique qui permettra l'assimilation de concepts par la pratique.

L'ampleur de nos productions sera limitée[9]; nous viserons à appliquer suffisamment les concepts explorés dans le cours pour être en mesure par la suite de comprendre les enjeux et de poser un jugement critique sur les expertises qui nous seront présentées.

Ce cours entraîne un problème pédagogique de fond. En effet, produire quelque chose de pertinent dans un cours d'une telle complexité demande qu'on se mette tôt à la tâche, avant même d'avoir commencé à aborder la matière.

La production de travaux se fera de manière concurrente avec la présentation de la matière. Cela tend à angoisser les étudiant(e)s qui, C'est légitime, aimeraient en savoir plus avant de commencer à travailler sur un projet concret.

Malheureusement, la complexité de la tâche fait en sorte que nos choix sont de nous limiter à un survol ou d'expérimenter en parallèle avec le cours. Je préfère une approche appliquée. Évidemment, l'évaluation des productions sera faite en connaissance de cause et tiendra compte de cette difficulté particulière et inhérente à la nature du cours.

Nous discuterons par contre de considérations propres à certains choix commerciaux importants pour une firme désireuse de produire ou d'exploiter des solutions CS, et nous prendrons soin de comparer, selon plusieurs critères, les principales approches de développement possibles pour des SCS. La motivation derrière cette démarche est de vous donner des outils pour poser un jugement sur les technologies CS à partir de leurs particularités, plutôt que de comparer des produits existants et risquant d'être désuets lorsque vous obtiendrez votre diplôme.

[8] La question de la sécurité, par exemple, sera omniprésente tout au long de la session. Il en sera de même pour les considérations de performance, de stabilité et de tolérance aux pannes, entre autres sujets.

[9] Faute de temps, nos systèmes seront tous assez petits en comparaison avec ceux du monde réel

Objectifs spécifiques

Au terme du cours, l'étudiant(e) sera capable :

{1} D'identifier les enjeux propres au développement d'un SCS contemporain.
{2} D'identifier les enjeux historiques et contemporains propres au déploiement et à la mise à jour d'un SCS.
{3} D'identifier les différences entre une solution de type CS et une solution monolithique.
{4} De reconnaître différents modèles CS : interne, externe local, externe distant, par appel direct, requête/ réponse, événementiel (par rappel, ou *Callbacks*), *etc*.
{5} D'implanter un modèle transactionnel synchrone ou asynchrone, par rappel ou non.
{6} De développer un client ou un serveur selon une spécification d'interface donnée.
{7} De comprendre le rôle des principaux schémas de conception utilisés dans les SCS.
{8} De comprendre les enjeux liés à la tolérance au cycle de vie d'un service.
{9} De comprendre les enjeux propres à la dynamique des systèmes répartis.
{10} De comprendre les implications d'un SCS sur la sécurité informatique.

Adresse électronique pour remise de travaux

Soumettez tout travail remis électroniquement à l'adresse Patrice.Roy@USherbrooke.ca

Médiagraphie

Des notes de cours vous seront distribuées, et devraient être votre référence principale pour la session qui s'annonce. Aucun manuel n'est obligatoire.

Le site Web du cours devrait être, avec les notes de cours en tant que telles, votre référence principale :

http://h-deb.ca/UdeS/SCS/

De manière générale, le site Web de l'enseignant peut vous être utile à plus d'un point de vue :

http://h-deb.ca/

Un ensemble toujours croissant de liens portant sur les SCS peut être consulté ici :

http://h-deb.ca/Liens/Client-Serveur--Liens.html

Un ensemble toujours croissant de liens portant plus spécifiquement sur les applications et les technologies Internet peut être consulté ici :

http://h-deb.ca/Sujets/Web/index.html

Une liste annotée de volumes portant sur les systèmes répartis peut être consultée ici :

http://h-deb.ca/Liens/Suggestions-lecture.html#systemes-repartis

Pour enrichir le volet multiprogrammation, vous pouvez jeter un coup d'œil à :

http://h-deb.ca/Sujets/Parallelisme/index.html

Enfin, une liste annotée de volumes portant sur la pratique de la programmation et sur l'architecture de systèmes peut être consultée ici :

http://h-deb.ca/Liens/Suggestions-lecture.html#pratique-programmation