Examen ISA*: Introduction to Software Architecture*

- Date : 25 mai 2016
- Durée : 14h00 – 17h00 (3 heures)
- **Aucun document autorisé; barème donné à titre indicatif.**

Lisez tout le sujet (3 pages) avant de commencer à répondre aux questions ; Les questions sont identifiées en gras dans le texte ; On compte habituellement dix mots pour une ligne ; Les différentes parties sont indépendantes.

Vous devez répondre aux questions posées dans le sujet sur la copie d'examen fournie, avec d'éventuelles feuilles intercalaires. Vous ne pouvez pas sortir de la salle d'examen durant la première heure (avant 15h00), ni durant le dernier quart d'heure (après 16h45). **Toute fraude identifiée sera systématiquement transmise au conseil de discipline de l'UNS**.

Partie #1 : Question de recul /3 (0h15)

Durant votre projet « Isola 3000 », vous avez été confronté au développement concret d'une architecture 3-tiers. En utilisant les connaissances acquises durant votre formation dans le département (vous pouvez faire référence à d'autres cours qu'ISA) et votre expérience en projet ISA, **répondez à la question donnée en annexe #1**. Votre réponse doit être synthétique (~150 mots, 200 max).

Partie #2 : Analyse de texte /4 (0h30)

L'annexe #2 contient un article anglophone sur les architectures 3-tiers. Sur la base de votre expérience en conception et mise en œuvre d'architecture 3-tiers acquise durant le semestre, **proposez une analyse critique** et synthétique de ce texte mettant en avant votre opinion argumentée (~150 mots, 200 max). L'identification de critères pertinents utilisés dans votre analyse fait partie de la question.

Partie #3 : Étude de cas /13 (2h15)

L'annexe #3 contient un *briefing client* du projet obtenu par votre société. Sur la base des informations contenue dans ce briefing, vous devez proposer une architecture 3-tiers y répondant. Les points suivants vous permettront de cadrer votre étude de cas :

1. Identifiez les objets métiers au sein de votre architecture ;
   - **Décrivez ces objets sous la forme d'un diagramme de classes** ;
   - **Expliquez le *mapping* Objet-Relationnel quand c'est pertinent**.
2. Identifiez les différents composants à mettre en jeu dans votre architecture ;
   - **Décrivez l'assemblage** sous la forme d'un diagramme de composants ;
   - **Décrivez les interfaces** de chaque composant (diagramme de classes).
3. **Justifiez vos choix de conception** pour cette architecture, en identifiant quels éléments appartiennent à quel tiers de l'application, et pourquoi ils sont nécessaires dans le système. Soyez synthétique : ~150 mots, 200 max ;
4. Expliquez **comment cette architecture s'implémente** dans l'écosystème J2E. Soyez synthétique : ~150 mots, 200 max.

Annexes

Annexe 1 : Thème pour la question de recul

« *Quels sont les apports d'une couche Web Services dans une architecture logicielle ?* »

Annexe 2 : Enterprise Application Architecture[1] (Martin Fowler)

Enterprise applications rarely live on an island. Usually they need to **integrate with other enterprise applications** scattered around the enterprise. The various systems are built at different times with different technologies, and even the collaboration mechanisms will be different: COBOL data files, CORBA, messaging systems. Every so often the enterprise will try to integrate its different systems using a common communication technology. Of course, it hardly ever finishes the job, so there are several different unified integration schemes in place at once. This gets even worse as businesses seek to integrate with their business partners as well.

Then there's the matter of what comes under the term "business logic." I find this a curious term because there are few things that are less logical than business logic. When you build an operating system you strive to keep the whole thing logical. But business rules are just given to you, and without major political effort there's nothing you can do to change them. You have to deal with a haphazard array of strange conditions that often interact with each other in surprising ways. Of course, they got that way for a reason: Some salesman negotiated to have a certain yearly payment two days later than usual because that fit with his customer's accounting cycle and thus won a couple of million dollars in business. A few thousand of these one-off special cases is what leads to the **complex business "illogic"** that makes business software so difficult. In this situation you have to organize the business logic as effectively as you can, because the only certain thing is that the logic will change over time.

For some people the term "*enterprise application*" implies a large system. However, it's important to remember that **not all enterprise applications are large**, even though they can provide a lot of value to the enterprise. Many people assume that, since small systems aren't large, they aren't worth bothering with, and to some degree there's merit here. If a small system fails, it usually makes less noise than a big system. Still, I think such thinking tends to short-change the cumulative effect of many small projects. If you can do things that improve small projects, then that cumulative effect can be very significant on an enterprise, particularly since small projects often have disproportionate value. Indeed, one of the best things you can do is turn a large project into a small one by simplifying its architecture and process.

[1] Extrait de « *Patterns of Enterprise Application Architecture* » (dans la bibliographie du cours), pages 3-4.

Annexe 3 : Étude de cas *UltraPlanning*

La société *MajeurEducation* souhaite faire évoluer sa plateforme de gestion d'emploi du temps et de planning. Le nouveau produit s'appellera « *UltraPlanning* ». Par rapport a son prédécesseur, *UltraPlanning* permettra une meilleure gestion des emplois du temps en permettant le positionnement automatique des cours en fonction des contraintes temporelles.

Le système doit permettre de réaliser les actions suivantes :

- Un administrateur déclare les directeurs de chaque département de l'école ;
- Chaque directeur de département (ou l'administrateur) déclare les responsables de promotions (année, ou parcours);
- Le catalogue des salles disponibles est saisi par le secrétariat de l'école ;
- Certaines matières nécessitent des salles particulières, par exemple dans le département Génie Biologique ou en Électronique. Ces contraintes sont saisie par le secrétariat ;
- Le responsable d'une promotion déclare des cours, associés à des enseignants. Il affecte à chaque cours des séances (cours en amphi, séance de TD, régularité) et groupes d'étudiants (qui peuvent être partagés par plusieurs cours).
- Une fois toutes les séances entrées, le responsable d'année peut demander le placement automatique de son emploi du temps à *UltraPlanning.* Le système affecte automatiquement une salle et un horaire a chaque séance. Il retourne au responsable d'année la liste des séances qu'il n'arrive pas à placer ;
- Les responsables (et directeurs de département ou administrateurs) peuvent déplacer des séances à une date donnée. Le système affecte automatiquement une salle si celle ci n'est pas donnée lors du déplacement ;
- Les étudiants peuvent demander une modification d'emploi du temps ponctuelle via le système. Le responsable d'année peut la valider, ou la refuser en expliquant pourquoi.
- Lorsqu'une modification d'emploi du temps est effectuée a courte échéance (moins de 48h), un système externe de notification email est appelé pour prévenir les personnes concernées du changement (nouvelle salle, horaire, …)