Projet
Architectures et Styles d'Architectures

Alexandre Mélo, Glenn Plouhinec

1 En quoi consiste ASA ?

ASA est l'abréviation de Architectures et Styles d'Architectures. Cette matière a plusieurs objectifs à nous faire réaliser. Ces objectifs sont étroitements liés à notre projet. L'objectif le plus important (même si tous les objectifs sont importants) est la compréhension des concepts clés des architectures logicielles. Le deuxième objectif est le fait d'être capable d'identifier et de développer différents styles architecturaux. Et un autre objectif qui est très important et que nous avons beaucoup vu avec ce projet, est le fait d'être capable de maîtriser la complexité des architectures logicielles à un haut niveau d'abstraction couplé au fait de comprendre les principales difficultés qui interviennent lors du passage à l'échelle. Ce projet est un concentré de tous ces objectifs et a pour but de résumer une bonne partie de ce que la matière nous propose. Ceci, est notre projet.

2 Projet

2.1 M2

2.1.1 Spécification

Le niveau de modélisation *M2* est le méta-modèle permettant de définir le langage que l'on souhaite utiliser pour un domaine d'utilisation précis. Notre méta-modèle nous permet de définir un langage pour les architectures logicielles, et les styles architecturaux. Le méta-modèle que nous avons défini a été conçu grâce aux plugins EMF et Ecore, nous permettant de représenter sous la forme de classes les différents concepts des architectures logicielles.

2.1.2 Configuration et éléments architecturaux

Pour concevoir ce méta-modèle, nous nous sommes basés sur la syntaxe du langage *ACME* `https://www.cs.cmu.edu/~acme/docs/language_overview.html`. Dans l'exemple, ci-dessous, on remarque déjà qu'une *Configuration* est composée de divers éléments architecturaux: *Component*, *Connector*, et *Attachment*. Les *Components* sont composés de *Ports*, les *Connectors* ont des *Roles*, et les *Attachments* font le lien entre les *Ports* et les *Roles*.

```
System SimpleCS = {
    Component Client = { Port Send_Request }
    Component Server = { Port Receive_Request }
    Connector RPC = { Roles { Caller, Called } }
    Attachements : {
        Client.SendRequest to RPC.caller,
        Server.Receive_Request to RPC.called
    }
}
```

De plus, en nous appuyant sur nos connaissances du cours, et d'autres exemples du langage *ACME*, nous sommes capables de définir de nouveaux liens: un *Component* peut être constitué d'une *Configuration*, ce qui nous donnerai une structure arborescente lors de la modélisation de *M1*. Les structures arborescentes peuvent être modélisées grâce au patron de conception *Composite*. Nous avons donc utilisé ce patron pour débuter notre modélisation :

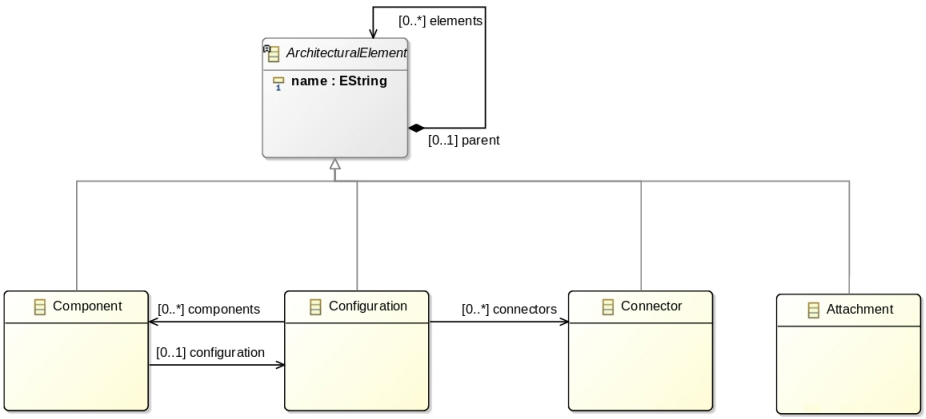

Figure 1: Patron Composite appliqué au méta-modèle

2.1.3 Interfaces

Les *Configurations*, les *Components*, et les *Connectors* disposent d'interfaces pour pouvoir communiquer avec d'autres éléments architecturaux. Un *Component* n'existe que s'il possède un *Port* et un *Service* fournis, une *Configuration* peut avoir des *Ports* fournis ou requis, mais ne peut pas disposer de *Services*:

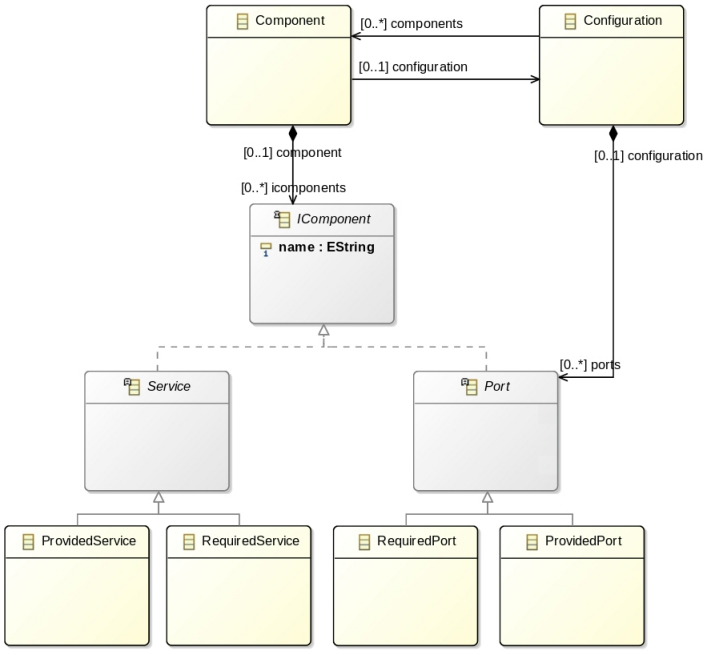

Figure 2: Interfaces: Ports et Services

Les interfaces des *Connectors* sont différentes. Ceux-ci ne disposent que de *Roles* fournis et requis, qui seront ensuite liés à des *Ports* grâce aux *Attachments*. Les *Connectors* sont également composés d'une *Glue* qui fait le lien entre les *Roles*.

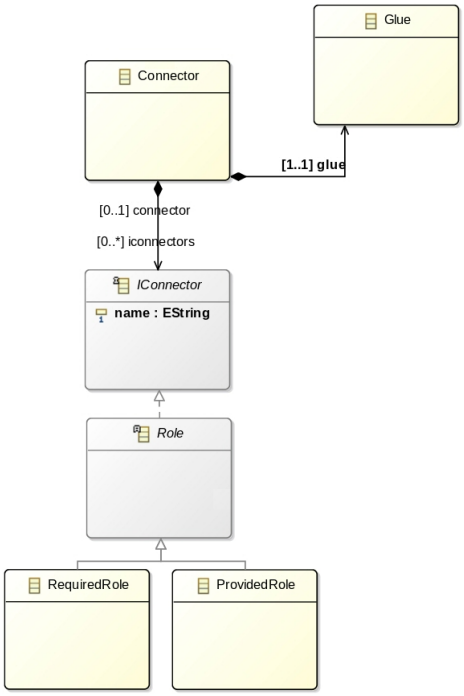

Figure 3: Interfaces: Roles

2.1.4 Liens

Il existe deux liens qui lient les différentes interfaces entre elles, et que nous avons choisi de modéliser:

- Les *Bindings* lient les ports d'une *Configuration* aux ports d'un *Component*
- les *Attachments* lient les rôles d'un *Connector* aux ports d'un *Component*

Nous avons choisi de restreindre les cardinalités de ces liens car nous voulions respecter la sémantique des exemples étudiés: un *Attachment* lie un *ProvidedPort* à un *RequiredRole*, et/ou un *RequiredPort* à un *ProvidedRole*, et un *Binding* lie un *ProvidedPort* à un *RequiredPort*. Nous avons ainsi défini les cardinalités *[1...2]* pour les rôles et ports de *Attachment*, et *[2...2]* pour les ports de *Binding*.

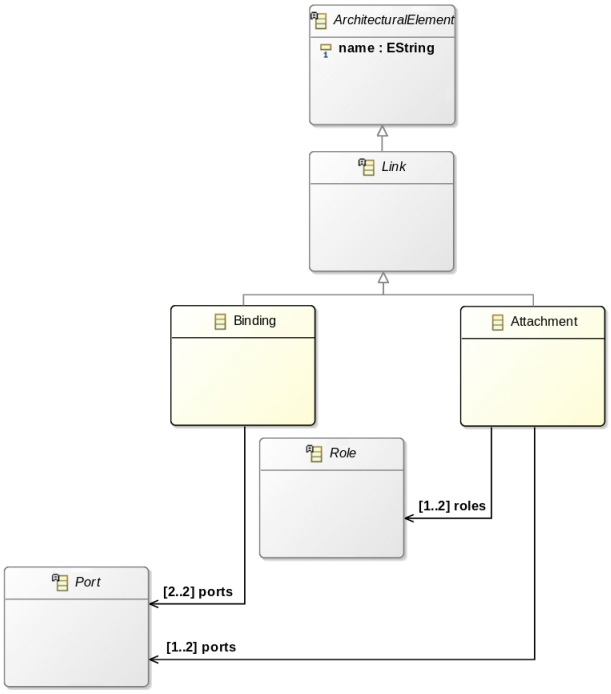

Figure 4: Binding et Attachment

Le diagramme complet de *M2* se trouve dans le fichier *model/asa.ecore#asa.M2*, ou via l'image *model/M2.jpg*.

2.2 M1

2.2.1 Spécification

Le niveau de modèlisation *M1* est un niveau d'abstraction moins élevé que *M2*, il consiste à modéliser un système réel défini par le langage de *M2*. Le modèle réel que nous concevons dans cette partie est une architecture Client-Serveur. La modélisation de M1 a été faite de plusieurs manières:

- Nous avons tout d'abord créé une instance dynamique de *M2* dans Ecore grâce à un point d'entrée que nous avons ajouté à ce dernier, et nommé "*ArchitecturalSystem*".
- Un nouveau package Ecore a été créé, et nous avons créé de nouvelles classes qui héritent des classes de M2, notamment pour faciliter la génération de code, pour le niveau de modélisation *M0*.
- Un programme en *ACME* a été écrit pour simplifier la compréhension du modèle *M1*, et les interactions entre les différents éléments architecturaux.

2.2.2 Présentation de l'approche

Grâce au plugin EMF et les outils fournis avec Ecore, il est possible de créer un langage correspondant aux "instances" **.xmi* du méta-modèle *M2*, avec *Xtext*. Ces "instances" correspondent au modèle *M1*. Il est également possible de générer du code Java à partir de ces fichiers **.xmi* avec *Xtend*, pour créer facilement le niveau de modélisation *M0*.

Le langage correspondant est *ACME*, il permet de donner une syntaxe concrète textuelle à notre fichier *clientserver.xmi*. Il est également possible de créer une syntaxe concrète graphique avec *Sirius*. La syntaxe concrète a pour objectif d'améliorer la compréhension et la lisibilité du modèle. Dans notre approche, nous souhaitions définir un modèle *clientserver.xmi*, avoir une syntaxe concrète textuelle correspondant à ce même modèle sous la forme d'un programme *ACME*, et générer du code Java grâce à Xtend. Malheureusement, Xtend n'a jamais pu fonctionner sur nos machines, et nous avons dû opter pour une autre approche pour générer du code Java correspondant au système client-serveur: la création de nouvelles classes Ecore, qui étendent le méta-modèle *M2*.

2.2.3 Modélisation du système Client-Serveur

Le modèle Client-Serveur est défini dans le fichier *model/clientserver.xmi*, et son programme *ACME* dans *model/clientserver.acme* dont voici un extrait:

```
Configuration Client_Server = {
  Component Client = {
    ProvidedPort Send_Request;
    RequiredPort System_Client;
  }

  Component Server = {
    Configuration Server_Detail = {...}
    RequiredPort Receive_Request;
    RequiredPort System_Server;
    ProvidedPort System_Server_Binding_Send;
  }

  Connector RPC = {
    RequiredRole Caller;
    ProvidedRole Called;
  }
  Attachment Client_To_RPC = {
    ProvidedPort: Client.Send_Request to RequiredRole: RPC.Caller;
  }
  Attachment RPC_To_Server = {
    RequiredPort: Server.Receive_Request to ProvidedRole: RPC.Called;
  }
  Binding Server_System = {
    ProvidedPort: Client_Server.Server_Port to RequiredPort: Server.System_Server;
  }
  Binding Client_System = {
    ProvidedPort: Client_Server.Client_Port to RequiredPort: Client.System_Client;
  }
  ProvidedPort Server_Port;
  ProvidedPort Client_Port;
}
```

Nous avons pris des libertés en adaptant la syntaxe *ACME* de manière à la rendre plus compréhensible pour notre cas d'utilisation, la différence entre les interfaces fournies et requises étaient au départ difficiles à comprendre.

2.2.4 Préparation à la génération de code

Le package Ecore *asa.M1* situé dansle fichier *model/asa.ecore* reprend globalement les mêmes éléments architecturaux que le modèle précédent. Un problème lors de la conception de ce diagramme Ecore est que les relations du métamodèle ne sont pas utilisées, mais de nouvelles relations sont créées. Par exemple la relation de composition entre une *Configuration* et un *Component*, si on cherche à relier les deux nouvelles classes *Configuration_ ClientServer* et *Component_ Client*, le code généré aura une nouvelle

relation plutôt que d'utiliser celles du méta-modèle. Cela ne sert alors à rien d'essayer de les relier, nous devrons compléter le code Java à la main pour faire le lien entre les différentes classes.

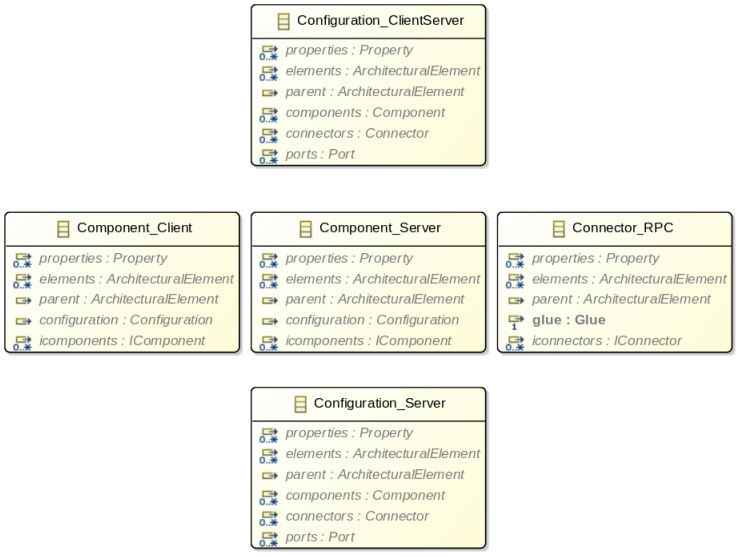

Figure 5: Extrait du diagramme M1

Le diagramme complet de *M1* se trouve dans le fichier *model/asa.ecore#asa.M1*, ou via l'image *model/M1.jpg*.

2.3 M0

2.3.1 Spécification

M0 est le niveau de modélisation "final" qui représente le système réel, conforme à *M1* et *M2*. Il s'agit pour nous du code Java correspondant à l'architecture Client-Serveur issue de *M1*.

2.3.2 Implémentation

Le code généré de M1 et M2 situé dans *src-gen/* a été un peu modifié pour nos besoins. Le code Java qui nous intéresse particulièrement se trouve dans le répertoire *src/*.

3 Discussion

Certains aspects du modèle restent incomplets ou ambiguës, par exemple le fait que nos *Connectors* n'aient que deux rôles, pour qu'une communication à double sens soit effective, faudrait-il autant de *RequiredRoles* que de *ProvidedPorts* ? Et autant de *ProvidedRoles* que de *RequiredPorts* ? Le problème s'est posé pour la configuration *Server_Detail* : nous avons voulu modéliser une communication à double sens pour chacun des composants, on l'on peut envoyer un message (répondre à une requête), et en recevoir un. Dans notre modèle, tous les composants envoient leur message sur le même *RequiredRole* du connecteur qui leur correspond. Le connecteur doit alors savoir vers quel *RequiredPort* retransmettre le message à partir du *ProvidedRole*. Cette opération serait sans doute plus simple si chaque rôle du connecteur était relié à un unique port d'un composant. Faute d'informations à ce sujet, nous avons gardé des connecteurs à deux rôles.

Nous avons également la notion de *Property* qui n'a pas été exploitée, l'intérêt du projet s'étant porté sur les relations entre les configurations, composants, connecteurs, attachments et bindings, nous aurions pu approfondir d'autres aspects.

Nous aurions aussi pu ajouter des contraintes sur le méta-modèle, par exemple, une contrainte *OCL* qui empêcherait le fait qu'un *Link* ou un *Connector* n'ait d' *ArchitecturalElements* (relation issue du patron Composite), ou encore spécifier le fait qu'un *Binding* ou un *Attachment* fasse le lien entre une interface requise et une interface fournie. Si un méta-modèle est complet et respecte toutes les spécifications, la modélisation de *M1* et *M0* se fait beaucoup plus facilement.

Une façon d'améliorer le projet en plus de ce qui est abordé juste avant est le fait de gérer plusieurs serveurs mais que ces serveurs communiquent aussi entre eux. Ce qui nous intéresserait le plus c'est de pouvoir utiliser Xtend pour générer notre M1 en code java et ainsi pouvoir simplifier un maximum l'implémentation de M1 en code.

4 Conclusion

Dans ce projet, nous avons pu aborder et comprendre les concepts clés des architectures logicielles, nous avons eu l'occasion de développer une ébauche d'une architecture client-serveur. Nous avons notamment appris comment mieux utiliser les outils *EMF* et *Ecore*, et les principes de modélisation qui en découlent, avec les niveaux de modélisation *M2*, *M1*, et *M0*. Le méta-modèle *M2* que nous avons conçu n'est pas une solution unique, et des travaux supplémentaires permettraient d'améliorer ce dernier, notamment avec l'ajout de contraintes, pour respecter le plus possible les spécifications.