Chapitre 7

Architecture et qualité de systèmes logiciels

Nicole Levy[1], Francisca Losavio[2], Yann Pollet[1]

Résumé

Définir l'architecture d'un système logiciel est une étape essentielle de son développement. C'est la première étape de prise de décisions de conception. C'est en particulier là que seront prises les décisions concernant la manière de prendre en compte les exigences non fonctionnelles dont celles de qualité. Lorsqu'il s'agit de construire un nouveau système, il est intéressant de valoriser la connaissance acquise dans le développement d'autres systèmes comparables, c'est-à-dire appartenant au même domaine fonctionnel. Notre propos est d'étudier comment la capitalisation de la connaissance afférant à un domaine fonctionnel donné peut permettre d'atteindre l'objectif recherché, à savoir une architecture de qualité. Nous proposons une démarche de construction d'une architecture guidée par la qualité et appliquons cette démarche dans un premier temps pour le développement d'une architecture de référence d'un domaine fonctionnel puis pour celle d'un système concret. Nous développons une étude de cas réelle dans le domaine de la santé.

7.1. Introduction

Définir l'architecture est une étape essentielle du développement d'un système logiciel. C'est la première étape de prise de décisions de conception. C'est là que sera décidée la décomposition en composants et la manière dont ils communiqueront. Une mauvaise décomposition aura des répercussions sur le produit qu'il sera très difficile ou couteux de corriger par la suite. Aussi, il est important, lors de cette étape, d'avoir des outils d'évaluation de la qualité future du système et de pouvoir

[1] Laboratoire CEDRIC, Cnam, Paris, France
[2] *Laboratorio MoST, Centro ISYS, Escuela de Computación, Facultad de Ciencias, Universidad Central de Venezuela, Caracas, Venezuela*

faire des choix en connaissance de cause. S'il n'est pas possible d'évaluer a priori une architecture de manière absolue, il est par contre possible de la comparer avec d'autres solutions envisageables.

L'objectif est d'aller vers la meilleure architecture possible, celle qui répondra au mieux aux besoins fonctionnels et non fonctionnels exprimés dans l'analyse des besoins. Différentes solutions existent pour guider le développement d'une solution architecturale. La plupart des méthodes préconisent une démarche descendante, par décomposition. Mais une telle démarche est difficile à mener de bout en bout. L'architecte part rarement de rien, il réutilise de manière plus ou moins explicite une démarche, des développements existants, des systèmes proches, des composants.

Dans un domaine donné (par exemple le domaine bancaire ou celui de la santé), de nombreux systèmes assurent des missions similaires, et fournissent pour ce faire des fonctionnalités très proches, voire identiques d'un contexte à l'autre. On qualifie ici un tel ensemble de « domaine fonctionnel ». Cette notion est plus restrictive que celle plus classique de « domaine », où les fonctionnalités des systèmes informatisés à construire peuvent s'avérer très diverses ; par exemple pour le domaine « santé », « dossier médical partagé » est un domaine fonctionnel.

Lorsqu'il s'agit de construire un nouveau système, il est intéressant de valoriser la connaissance acquise dans le développement d'autres systèmes comparables, c'est-à-dire appartenant au même domaine fonctionnel. Ceci concerne en particulier les fonctionnalités à assurer, les exigences de qualité communes à tous les systèmes, ainsi que les choix de conception qui en ont résulté. Les méthodes actuelles de développement d'architecture ne permettent pas de le faire, et proposent essentiellement des démarches de développement ex nihilo [HOF07], [RAS11]. L'expérience montre cependant que des fonctionnalités proches se traduisent, d'une part, par des choix architecturaux réutilisables dans d'autres contextes, et d'autre part, par des différences tout à fait significatives dans les solutions à mettre en œuvre du fait d'une part des fonctionnalités effectives à assurer, mais aussi d'autre part, des exigences non fonctionnelles propres au contexte spécifique.

Notre propos est d'étudier comment la capitalisation de la connaissance afférant à un domaine fonctionnel donné peut permettre d'atteindre l'objectif recherché, à savoir une architecture de qualité. Pour cela nous allons présenter en section 7.2 la notion de qualité dans les architectures, puis en section 7.3 la démarche proposée de construction d'une architecture guidée par la qualité. Nous appliquerons cette démarche dans un premier temps en section 7.4 pour le développement d'une architecture de référence d'un domaine fonctionnel puis en section 7.5 pour celle d'un système concret. Nous développons une étude de cas réelle dans le domaine de la santé pour illustrer notre démarche globale. La section 7.6 décrit des travaux connexes.

7.2. La démarche qualité

7.2.1. *Le standard de qualité ISO 25010*

Étudier la qualité du logiciel c'est se concentrer sur l'étude des facteurs ayant un impact sur la qualité des produits logiciels et de la capacité d'utilisation de ces produits dans un contexte particulier [CAL97]. La qualité est généralement définie en termes de conformité aux spécifications et d'adéquation à l'utilisation visée ([CRO84], [DEM00], et beaucoup d'autres). Cependant, cette définition a ses limites. Par exemple, dans le cas d'une mauvaise ou insuffisante spécification, le produit pourra être conforme à sa spécification sans satisfaire à la qualité requise; en ce qui concerne l'adéquation à l'utilisation visée, un produit peut remplir sa fonctionnalité mieux qu'un autre. A titre d'exemple, nous citerons ici deux définitions tirées des normes bien connues IEEE [IEE83] et ISO/IEC 25030 [ISO06].

Selon la norme IEEE, la qualité du logiciel est l'ensemble des propriétés et caractéristiques d'un produit logiciel qui portent sur sa capacité à satisfaire des besoins donnés, comme être conformes à des spécifications. Selon la norme ISO/IEC 25030, la qualité du logiciel est la capacité du produit logiciel à satisfaire des besoins exprimés et implicites lorsqu'il est utilisé dans des conditions spécifiées.

La norme ISO/IEC 9126-1 et sa mise à jour ISO/IEC 25010 [ISO11], qui fait partie de la série de normes SQuaRE [ISO06], par le « *Joint Technical Committee ISO/IEC JTC 1, information technology, Subcommittee SC 7, Software and Systems Engineering* », définissent des propriétés de qualités qui peuvent être utilisées pour décrire un produit logiciel. Les propriétés de qualité sont représentées par un modèle de qualité constitué d'une hiérarchie de caractéristiques basée sur 8 caractéristiques principales (figure 7.1) : adéquation fonctionnelle (« *Functional suitability*»), performance (« *Performance efficiency* »), compatibilité (« *Compatibility* »), facilité d'utilisation (« *Usability* »), fiabilité (« *Reliability* »), sécurité (« *Security* »), maintenabilité (« *Maintainability*») et portabilité (« *Portability* »). Ces caractéristiques, dites abstraites, sont ensuite raffinées jusqu'à atteindre des caractéristiques mesurables appelées attributs de qualité. Par exemple, l'adéquation fonctionnelle implique l'existence d'un ensemble de fonctions qui satisfont un besoin particulier. Il est raffiné en sous caractéristiques telles que complétude (« *Completeness* »), correction (« *Correctness*») et pertinence (« *Appropriateness*»). Ces sous caractéristiques peuvent elles-mêmes être raffinées selon le domaine d'application visé. La hiérarchie de caractéristiques de la norme ISO/IEC 25010 constitue le modèle de qualité d'une application ou d'un logiciel dans un domaine donné. Il faut noter qu'un produit logiciel peut aussi être un sous-produit du développement, par exemple, un modèle de cas d'utilisation ou un modèle d'architecture.

En ce qui concerne la qualité de l'architecture logicielle, les caractéristiques de qualité des logiciels (souvent appelés attributs de qualité dans la littérature alors que dans la norme ISO/IEC 25010, la notion d'attribut est restreinte aux caractéristiques mesurables) peuvent être considérés comme des points de référence décrivant le comportement attendu du système dans l'environnement pour lequel il est construit. Il s'agit de normes en fonction desquelles les résultats peuvent être mesurés ou évalués. Les caractéristiques de qualité, spécifiées par un modèle de qualité, peuvent décrire la manière de mesurer l'adéquation de l'architecture d'un système en cours de développement, et permettent par extrapolation d'en déduire quelle sera la qualité du système entier. L'architecture logicielle a, d'une manière ou d'une autre, un impact déterminant sur la plupart des caractéristiques de qualité des systèmes. Cependant, il n'existe pas de règles permettant de préciser les besoins de qualité directement impactés par l'architecture. C'est l'expertise de l'architecte logiciel qui de par sa connaissance du domaine, joue un rôle important dans la définition du modèle de qualité associée à l'architecture.

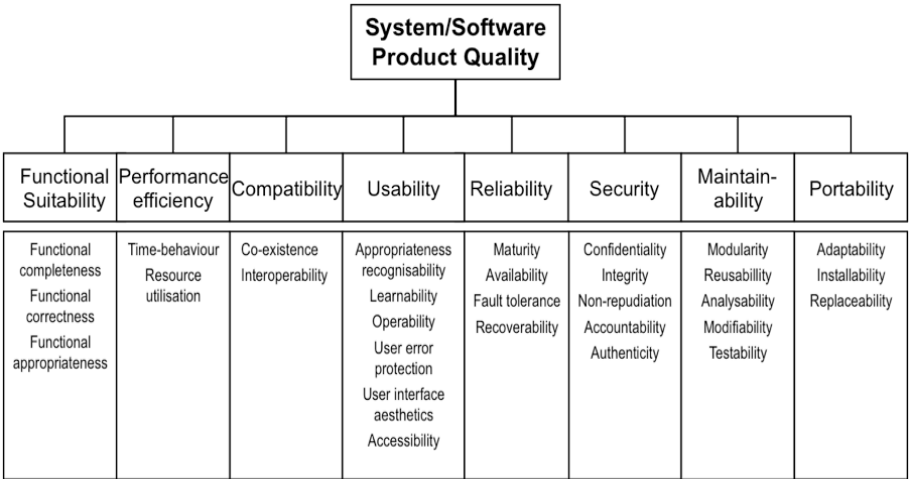

Figure 7.1. *Modèle de Qualité logicielle selon le standard 'ISO/IEC 25010 [ISO11]*

En général, la qualité d'un système logiciel dépend de la qualité des éléments qui le composent et de leurs interactions. Celle-ci peut être mesurée de manière interne, pendant la construction du système ou de manière externe lorsque le système est testé avant la livraison, et, enfin, mesurée par l'utilisateur final lui-même lorsque le système est en cours d'utilisation. Les propriétés du logiciel peuvent être des propriétés inhérentes au système, que celui-ci possèdera toujours mais dont la valeur peut changer si le système évolue. Il peut s'agir également de propriétés assignées qui peuvent changer sans modifier le logiciel (voir figure 7.2). Dans ce cas, il s'agit en général de propriétés requises par le client. Les besoins systèmes sont fixés par

l'ingénieur logiciel développant le système logiciel et concernent plutôt la configuration requise.

Software Properties	Inherent properties	Domain-specific functional properties
		Quality properties (functional suitability, reliability, performance efficiency, usability, security, compatibility, maintainability, portability)
	Assigned properties	Managerial properties (price, delivery date, product supplier ...)

Figure 7.2. *Classification des propriétés des logiciels [ISO06]*

Selon la classification SQuaRE, les exigences de qualité des logiciels représentent les besoins non fonctionnels (NFR) indiqués par le client en tant que propriétés du logiciel, en même temps que les besoins fonctionnels (FR). Les deux types de besoins sont considérés comme des propriétés inhérentes aux besoins du logiciel. La qualité d'un produit logiciel dans un contexte d'utilisation particulier est déterminée par ses propriétés intrinsèques. Les propriétés fonctionnelles spécifiques au domaine concernent généralement la transformation des données d'entrée en données de sortie. D'autres fonctions ou des fonctionnalités implicites peuvent être nécessaires pour mettre en œuvre les besoins en qualité et les contraintes nécessaires à l'accomplissement d'une fonctionnalité. Les besoins de qualité sont également associés aux besoins fonctionnels. Une caractéristique de qualité est le terme utilisé par la norme ISO/IEC 25010 modèle de qualité [ISO11] (voir figure 7.1), qui indique une exigence de qualité du logiciel qui peut être raffinée jusqu'à ce que les attributs de la qualité ou des éléments mesurables soient atteints. Les mesures à utiliser pour les exigences de qualité internes, externes et d'usage sont également fournies par la présente norme. Elles sont définies dans les normes ISO/IEC 9126-2, 3 et 4 et elles n'ont pas été modifiées dans la récente mise à jour de la norme. Dans ce travail nous prenons en compte seulement la qualité interne définie par [ISO11] pour mesurer les produits logiciels intermédiaires ou artéfacts, développés pendant le processus de développement d'un système ; l'architecture logicielle est un de ces produits et les mesures à ce niveau sont très abstraites. Par exemple, une valeur booléenne indiquera la présence d'un mécanisme ou la définition de mesures pour la performance et la fiabilité, qui dépendent de la complexité de la configuration architecturale en termes de composants et connecteurs [LOS03].

7.2.2. *Le référentiel de qualité*

Garantir la qualité d'un système se fait en étudiant à chaque étape du développement ce qu'il en est. Lors de l'analyse des besoins, les propriétés non fonctionnelles du futur système sont décrites : le modèle de qualité du système est

défini. Lors de la conception de l'architecture, il est important de vérifier que chaque critère de qualité requis sera bien pris en compte lors du développement du système : le modèle de qualité de l'architecture est alors défini. Nous allons étudier ces deux modèles de qualité en étudiant comment le deuxième raffine le premier et décrit comment devra être évalué le futur système pour vérifier la satisfaction des propriétés requises.

Dans l'analyse des besoins, les propriétés non fonctionnelles attendues décrites doivent être analysées pour leur associer une caractéristique de qualité associée au standard utilisé, ISO 25010 par exemple. Dans l'analyse des besoins, les propriétés non fonctionnelles attendues sont des restrictions concernant le futur système et son processus de développement ainsi que des contraintes externes (légales, économiques, technologiques, etc. …) que le système doit vérifier [KOT98].

Le standard ISO 25010 [ISO06] considère les propriétés inhérentes au produit : propriétés fonctionnelles liées au domaine et propriétés de qualité, ainsi que les propriétés assignées qui sont des propriété externes qui n'auront pas d'impact direct sur le produit. Notre objectif est d'étudier quels sont les besoins dont la prise en compte a un impact sur le développement de l'architecture. Nous n'allons ne considérer que ceux-là. Ce sont ceux qui doivent être pris en compte par l'architecture pour pouvoir garantir leur satisfaction par le futur système. Il s'agit donc des besoins qui sont liés au produit : le futur système. Il s'agit souvent de contraintes concernant son comportement, par exemple concernant ses performances ou ses capacités. Ces propriétés peuvent être facilement quantifiables et une mesure ainsi qu'une valeur peuvent être précisées dans l'analyse des besoins. Elles auront un impact direct sur la manière de réaliser le système. D'autres propriétés non fonctionnelles sont moins facilement quantifiables, comme l'utilisabilité, la maintenabilité ou la réutilisabilité. L'analyse des besoins peut être très vague à ce sujet ou préciser ce qui est entendu par ces termes et à quel niveau d'abstraction ils sont utilisés ; d'autre part, si la norme l'ISO/IEC 25010 est utilisée, ces termes sont définis dans le standard. Dans ce cas, l'analyse précisera la manière dont sera évaluée la prise en compte de cette propriété. Que ce soit dans l'analyse des besoins ou après, il faudra de toutes manières le préciser. De ces propriétés doit être dérivé le modèle de qualité du système. Nous allons étudier ce modèle à présent.

7.2.3. *Modèle de qualité d'un système*

Le modèle de qualité d'un système décrit ses besoins en termes de qualités. Nous considérons le modèle ISO 25010 et proposons d'utiliser un tableau où pour chaque caractéristique et sous-caractéristique du standard, est précisé comment celle-ci doit être considérée dans le système : les besoins la mentionnant ; une priorité assignée (par exemple sur une échelle de 5, la priorité la plus forte étant de 1); la

manière de l'évaluer (une mesure) ; un objectif valué fixé pour la mesure (voir tableau 7.1). Prenons comme exemple les besoins suivants :

[i] Le système devra répondre à la requête « recherche de disponibilité pour un vol précis » en moins de 1s ;

[j] Le système devra répondre à la requête « recherche des sièges disponibles dans un vol précis» en moins de 1s ;

[k] l'utilisateur devra pouvoir effectuer sa demande de recherche de disponibilité pour un vol précis en moins de 3 m.

Le modèle de qualité dérivé à partir de ces besoins est décrit par le tableau 7.1. Remarquons que parfois, la mesure peut ne pas être indiquée ou l'être très vaguement dans l'énoncé du besoin. Par exemple, il sera indiqué que le système devra pouvoir résister à des attaques extérieures sans plus de précision, ou que le système devra répondre à telle ou telle norme. Dans ce cas, l'architecture devra indiquer de quelle manière ces besoins auront été compris et pris en compte. Ces informations seront précisées dans le modèle de qualité de l'architecture.

Quality characteristics / Sub characteristics	**Priority**	**Requir ement**	**Measured Quality Attribute**	**Valued Objective**
Performance efficiency / Time behaviour	1	[i], [j]	Response time	1s, 1s
Usability / Operability	2	[k]	Time taken by the user to write his request	3m

Tableau 7.1. *Modèle de qualité représenté dans un tableau*

7.2.4. Modèle de Qualité Fonctionnelle

Le Modèle de Qualité Fonctionnelle (FQM) permet de relier les exigences fonctionnelles et les propriétés de qualité qui peuvent être requises pour bien accomplir chaque fonctionnalité. Le tableau 7.2 montre un exemple de FQM pour le domaine m-Banque. Par souci de lisibilité, les attribués mesurés et l'ordre de grandeur attendu ne sont pas indiqués.

7.2.5. Modèle de Qualité d'une Architecture

Une architecture est une proposition de structuration du futur système en termes de composants et de connecteurs communicants entre eux [SG96]. Il est important d'expliquer comment cette construction prend en compte les besoins exprimés dans

Functional Requirements		**Quality characteristics / Sub characteristics**
Authenti-cation	Token Card+PIN	*usability (accessibility, operability, user error protection), reliability (availability), security (authenticity), efficiency (time behaviour-response time)*
	Barcode	
	Shortcode	
Financial Information Services	Data consulting & operations not involving payment	*usability (accessibility), reliability, security (integrity), efficiency (time behaviour-response time)*
Accounting	Transfer between bank accounts	*reliability (availability), security (integrity, authenticity, confidentiality), efficiency (time behaviour-response time), functional suitability (correctness)*
	Credit card transactions	*reliability (availability), security (integrity, authenticity, confidentiality), efficiency (time behaviour-response time), functional suitability (correctness)*
	Service payment	*reliability (availability), security (integrity, authenticity, confidentiality), efficiency (time behaviour-response time), functional suitability (correctness)*

Tableau 7.2 : *Le modèle de qualité fonctionnel (FQM) du domaine m-Banque [LOS13]*

le modèle de qualité du système. Il s'agit de déterminer les responsabilités : qui fait quoi. Le modèle de qualité d'une architecture (AQM) est présenté également par un tableau précisant pour chaque sous caractéristique comment elle est prise en compte par l'architecture. Il peut s'agir d'un composant ou d'un groupe de composants. Par exemple, pour la performance, celle-ci est souvent indiquée pour un scénario d'utilisation. Lors de son exécution, plusieurs composants seront concernés. Le tableau devra également indiquer avec plus de précision la mesure appliquée et la valeur dérivée de la valeur mentionnée dans le modèle du système. Dans le cas précédent de la performance, chaque composant devra savoir quelle devra être sa performance requise. Ce tableau permet de déterminer l'origine exacte des propriétés de qualité par rapport aux besoins non fonctionnels de l'architecture, depuis l'analyse des besoins jusqu'aux composants les prenant en charge, introduisant des liens de traçabilité qui facilitent la vérification. Il est nécessaire pour pouvoir vérifier que tous les besoins seront bien pris en compte par l'architecture, avant même de pouvoir évaluer s'ils seront bien satisfaits. Le tableau indique aussi au concepteur comment la qualité sera mesurée, ce qui l'aidera à définir comment la réaliser. Parfois, la présence d'un composant garantit la prise en compte d'une qualité. C'est le cas par exemple de la sécurité qui sera assurée par les modules de cryptage/décryptage de l'information ou par un pattern, une solution architecturale ou un style [BUS 96]. Dans ce cas la mesure est booléenne : présence d'un dispositif prenant en charge la propriété. Le tableau 7.3 montre un exemple de

AQM pour le domaine de la m-Banque ou Banque Mobile, pour accéder à toutes les transactions bancaires à partir d'un téléphone portable. Par souci de lisibilité, les attribués mesurés et l'ordre de grandeur attendu ne sont pas indiqués.

Requirements related to Architecture		**Solution or Architectural Style**	**Quality characteristics / Sub characteristics (AQM)**
The database server must be centralized		*Repository*	*Security (integrity), maintainability (modifiability-scalability), reliability (availability),*
The communication layer for data access must be expandable and secured		*event channel – proxy*	*Maintainability (modifiability-scalability), security (authenticity, confidentiality, integrity), efficiency (time behaviour)*
The communications layer must be implemented using Web Services		SOA	*Compatibility (interoperability), security (authenticity), functional suitability (appropriateness, correctness), reliability (availability), maintainability (modifiability-scalability), portability (adaptability)*
Transmission	*SMS*	*event channel*	*Security of channel (integrity, confidentiality), reliability of channels (availability)*
	Mobile Web		
	Mobile Client Application		
	Secure SMS		

Tableau 7.3 : *Modèle de Qualité Architecturale (AQM) du domaine m-Banque, présenté comme un ensemble de scénarios [LOS13]*

L'utilisation de normes est recommandée dans les livres répertoriant les meilleures pratiques de l'ingénierie logicielle. En effet, elles facilitent la communication entre les intervenants et la compréhension. Néanmoins, leur utilisation n'est pas une pratique courante, car les normes manquent souvent de guide d'utilisation. C'est le cas en particulier pour les normes concernant la qualité des logiciels, et le fait qu'il n'y a pas d'accord sur la terminologie. Dans la pratique, chaque domaine utilise ses propres définitions des besoins et des paramètres de qualité. Une ontologie permettant de comparer les différentes normes de qualité a été proposée dans [LOS09] et [JEA12].

7.3. Démarche de développement d'une architecture de domaine

Nous allons développer une démarche générale de développement d'une architecture prenant en compte les deux modèles FQM et AQM présentés ci-dessus.

7.3.1 *Principes généraux*

De manière générale, la démarche de construction d'une architecture de domaine prend pour base des exigences système (« *system requirements* »), qui comprennent :

– Des exigences fonctionnelles propres au domaine, reflétant les fonctionnalités à fournir par le futur système ;

– Des exigences non fonctionnelles (dont les « exigences de qualité »), portant sur la manière dont sont fournies ces fonctionnalités, sous les aspects de qualité du service rendu et des aptitudes particulières attendues globalement du futur système ;

– des exigences de conception, ensemble des contraintes que l'on impose, pour toute raison que ce soit, au futur système, en terme de conception, de moyen de réalisation, etc. ou encore par exemple d'interfaces avec des systèmes déjà existants. Il peut s'agir d'hypothèses sur le modèle d'architecture : type de composants actif, passif, types d'interactions : synchrones, asynchrones, …

La conception d'un nouveau système vise à la définition d'une architecture système satisfaisant de telles exigences. Elle comprend en particulier :

– L'expression de la configuration architecturale, que l'on peut figurer comme un graphe [LOS12] dont les sommets sont les constituants du système, et les arcs constituent les connecteurs supports des interactions entre composants ;

– Les définitions des différents constituants, comprenant les interfaces offertes par chaque constituant, ainsi que les différentes exigences, fonctionnelles et non fonctionnelles, qui devront être vérifiées par chaque composant individuellement.

Les liens entre les éléments de l'architecture ainsi construite et les exigences systèmes les justifiant assurent la **traçabilité** des choix de conception. Cette traçabilité est essentielle dans toute démarche de construction d'un système[3].

La démarche que nous proposons ici articule la construction d'une architecture de système, d'un domaine donné, en deux phases, la première propre au domaine considéré, et donc commune à tous les systèmes, la seconde spécifique du contexte de chaque nouveau système particulier à construire. Ces deux phases sont :

– La construction d'une « architecture de domaine », base commune proposée pour la réalisation ultérieure de l'ensemble des systèmes du domaine ;

---

[3] Il convient ici de noter qu'un constituant peut-être un composant logiciel élémentaire, c'est-à-dire directement réalisé par une équipe de développement logiciel, ou bien un sous-système logiciel, qui doit être à son tour décomposé en composants plus fins et donner lieu à la définition d'une architecture interne à ce sous-système particulier, et ainsi jusqu'à obtention de composants logiciels élémentaires. La démarche de construction de l'architecture système apparaît alors comme essentiellement récursive.

– L'adaptation de cette architecture de domaine à un nouveau système à construire, en tenant compte des exigences particulières afférant à sa spécification.

L'architecture de domaine est l'architecture d'un système « idéal » (abstrait, car en général non réalisé) qui satisferait l'ensemble des exigences communes à la classe des systèmes considéré. Par exemple, pour un dossier médical partagé (qui peut être rencontré dans de nombreux contextes allant du logiciel individuel d'aide au praticien médical, jusqu'à un système d'information hospitalier pour un grand établissement), on retiendra les services attendus (besoins fonctionnels) absolument communs et que l'on sait indispensables à tous ces systèmes (consulter un dossier patient, ajouter un événement médical à celui-ci, etc.) et les exigences non fonctionnelles communes à tous ces systèmes (sécurité, performance, etc.).

Parfois, une exigence portant sur un attribut de qualité sera présente dans tous les systèmes du domaine, mais sachant également que la valeur attendue de l'attribut différera a priori d'un système à l'autre. Dans le contexte des lignes de produits présenté au chapitre 5, il s'agit d'un point de variabilité (c'est à dire de différents choix possibles connus au niveau domaine mais qui devront être sélectionnés au niveau système [POH05]) non fonctionnelle [BAR10]. Ainsi, la démarche basée sur un modèle de qualité permet de connaître exactement la fonctionnalité à l'origine de l'exigence de qualité. Ainsi, au niveau domaine fonctionnel, un attribut sera caractérisé par un ordre de grandeur ou une propriété. Cette valeur devra être précisée au niveau système. Par exemple dans le cas d'un système de gestion de dossiers médicaux, un attribut mesuré sera la volumétrie supportée par le système en terme de nombre maximum de dossiers gérés ; on sait qu'une exigence de ce type sera prescrite dans tout système du domaine, mais avec différentes valeurs d'un système à l'autre. Dans ce cas, l'exigence est retenue comme exigence de niveau domaine, avec pour valeur un paramètre à définir pour chaque système particulier. Sa valeur n'étant pas déterminée au niveau domaine, l'exigence correspondante devra être prise en compte ultérieurement, au niveau « système ».

La construction de cette architecture de référence du domaine est établie préalablement. Elle constitue la base de la construction des architectures des systèmes concrets particuliers. Cette architecture n'est sensée être remise en cause ensuite que de manière exceptionnelle à des fins d'enrichissement.

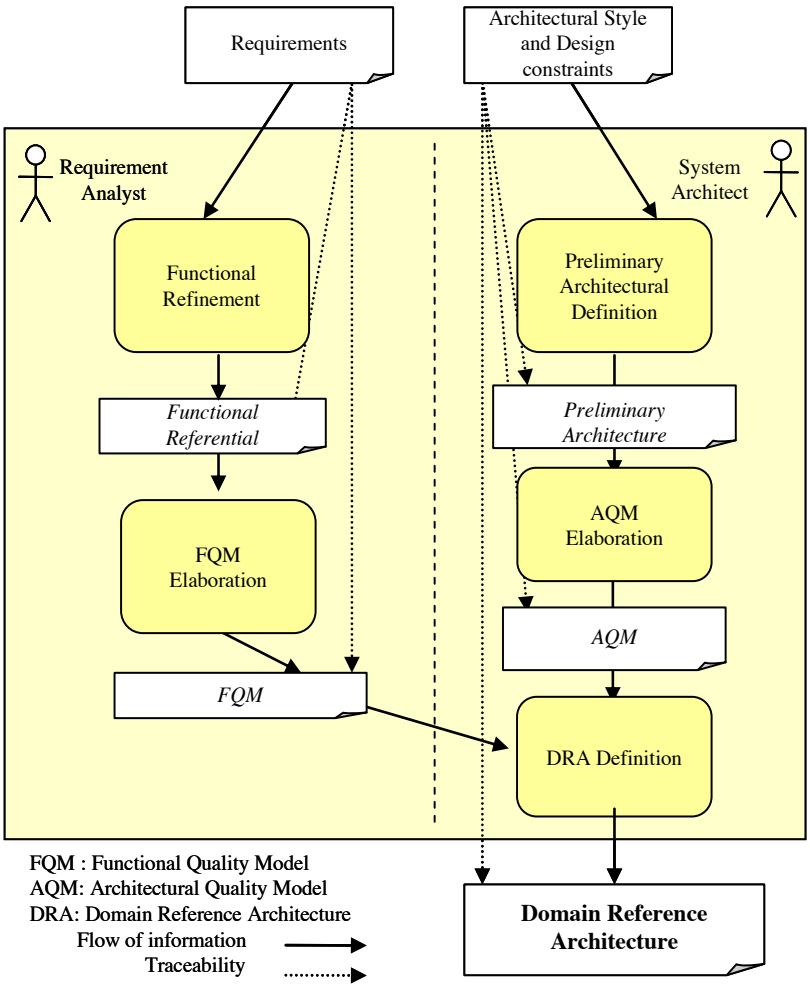

Figure 7.3. *Élaboration de l'Architecture*

L'élaboration de l'architecture de domaine d'une part, et de l'architecture d'un système particulier d'autre part, suivent la même démarche, dont le principe est symbolisé par la figure 7.3. La démarche considère en premier lieu deux ensembles d'activités menées selon deux branches parallèles. En ce sens, elle est proche de la démarche du cycle en Y [ROC01] mais si la première branche concerne dans les deux cas les aspects fonctionnels, l'autre n'est pas au même niveau d'abstraction puisqu'elle concerne les aspects techniques pour le cycle en Y et ici, l'architecture.

Les activités de la branche de gauche sur le schéma correspondent aux tâches menées par un analyste (« *Requirement Analyst* »), dont elles requièrent la

connaissance liée au domaine tandis que celles de la branche de droite requièrent les compétences d'un architecte système (« *System Architect* »).

7.3.2. *Le Modèle de Qualité Fonctionnelle*

L'activité de raffinement fonctionnel (« Functional Refinement ») analyse les besoins utilisateur en termes fonctionnels, et les formalise à l'aide d'un ensemble de fonctionnalités. Dans le cas du domaine fonctionnel, il faut les classer convenablement en fonctions nécessaires à tous les systèmes du domaine, ou bien seulement possibles, c'est-à-dire applicables seulement à des systèmes particuliers, en ne retenant que les premières

Partant des exigences utilisateur pertinentes, l'activité de définition de l'architecture préliminaire (« *Preliminary Architectural Definition* ») opère un choix de style architectural, et propose ainsi une première ébauche architecturale de système, base à des adaptations ultérieures.

7.3.3. *Le Modèle de Qualité Architecturale*

A partir de cette première version de l'architecture, l'activité nommée « *AQM Elaboration* » prend en compte les exigences non fonctionnelles non attachées à des fonctions particulières et s'appliquant globalement au système. Leur impact est analysé en termes d'architecture en les traduisant sous la forme d'adaptations de l'architecture précédente, et comme des assertions de qualité applicables à des composants de celle-ci.

7.3.4. *Architecture de Référence*

Les résultats issus de ces deux branches permettent conjointement l'élaboration de l'Architecture finale. Il s'agit selon la phase où l'on se situe, soit de l'Architecture de Référence propre au Domaine Fonctionnel, soit de l'Architecture du système particulier étudié. Rappelons que l'Architecture de Référence du Domaine, sert de base à l'élaboration des architectures spécifiques des nouveaux systèmes, par adaptations propres au contexte. Notons que la présentation de ces activités comme deux branches totalement indépendantes est une vue idéale, la pratique montrant, en général, des interactions entre les deux branches. Il importe bien sûr de maintenir les liens de traçabilité entre les assertions de qualité des différents modèles et celles dont elles sont dérivées.

Une fois élaborée l'Architecture de Domaine, la démarche suivie pour l'élaboration de l'architecture d'un système particulier à construire suit une

démarche tout à fait analogue à celle qui vient d'être décrite. Le point de départ en est les exigences utilisateurs spécifiques de ce nouveau système et l'architecture de référence issue de l'étude du domaine fonctionnel. Chaque activité devient une activité d'enrichissement des modèles dans le contexte des exigences propres du nouveau système pour obtenir les modèles de qualité applicables au système cible.

Principe de la démarche d'intégration architecturale : Pour construire une architecture adaptée aux exigences systèmes spécifiées, tant fonctionnelles que non fonctionnelles, la démarche adoptée consiste dans son principe à :

1) Partir de l'architecture « initiale » répondant aux exigences fonctionnelles, en relaxant l'ensemble des contraintes induites par les exigences de qualité. Il s'agit de l'architecture « idéale » qui traite le problème sans prendre en compte les exigences non fonctionnelles, c'est-à-dire sans se soucier des besoins en sécurité, des performances requises, etc. Une telle architecture n'est ainsi justifiée que par les exigences fonctionnelles et le(s) style(s) associé(s) au domaine. Le fait d'avoir un style implique qu'on a aussi toutes les exigences de qualité liées ou style.

2) Intégrer progressivement les différentes exigences de qualité a) en recherchant à chaque étape les différentes modalités possibles de prise en compte de l'exigence considérée dans l'architecture proposée à l'étape actuelle de la démarche, b) en choisissant l'une d'entre elles et en modifiant conséquemment l'architecture, et c) en vérifiant que l'architecture ainsi modifiée satisfait toujours l'ensemble des exigences de qualité considérées lors des précédentes étapes.

Les modifications de l'architecture introduites à chaque étape peuvent être :

– Des **ajouts de nouveaux composants**. Typiquement, un tel composant ajouté sera porteur d'une exigence fonctionnelle ou d'une exigence de qualité qui garantira l'obtention de la propriété de qualité recherchée au niveau système ;

– Des **ajouts d'exigences fonctionnelles ou non fonctionnelles** posées sur un composant déjà identifié et présent dans l'architecture ;

– Des **ajouts** d'exigences **concernant l'infrastructure d'exécution**;

– Des modifications **de la topologie de l'architecture** (connexions entre composants) impliquant un retour arrière pour vérifier la validité des choix préalables ;

– Des combinaisons de ces différentes possibilités.

Notons qu'au niveau domaine fonctionnel, les modifications peuvent être soit avoir impact direct sur l'architecture, soit avoir un impact différé. Contrairement à l'approche ligne de produits, tous les choix ne sont pas à priori connus, il sera toujours possible d'en définir un nouveau. Cette démarche est une heuristique justifiée par l'expérience. Elle ne garantit bien sûr pas l'obtention à coup sûr d'une solution, ni celle de la meilleure solution (meilleure au sens de moindre complexité). Elle est essentiellement itérative et peut nécessiter des retours arrière dans les

itérations, comme dans plusieurs démarches de conception d'architecture [BOS99], [KRU99], [ARA02], [BAS02a].

Enfin, il est par ailleurs clair que l'ordre de prise en compte des exigences de qualité influe à la fois sur le processus et possiblement sur le résultat. C'est pourquoi il est important de prendre les exigences de qualité dans un ordre qui conduise globalement aux remises en cause de moindres ampleurs. L'architecte doit donc pour ce faire déterminer un certain ordre de prise en compte des exigences de qualité. Cet ordre doit correspondre à un classement par ordre de difficulté décroissante de prise en compte, ou encore d'impact potentiel décroissant sur l'architecture. Ce classement repose bien sûr essentiellement sur le savoir faire de l'architecte et les références à des cas similaires issus de son expérience.

7.3.5. *Passage du niveau Domaine Fonctionnel au niveau Système*

Nous avons développé une démarche en deux étapes : la première s'applique au niveau domaine fonctionnel, la deuxième au niveau système.

Le niveau domaine fonctionnel répond à un besoin de réutilisation, tant des fonctionnalités caractéristiques d'un domaine que de la structure générale ou architecture de référence correspondante. Ce niveau est proche de celui considéré dans les approches lignes de produit, mais l'ensemble des solutions n'a pas été exploré, ni même défini ce qui rend le domaine fonctionnel plus ouvert ou apte aux évolutions. Il s'agit d'une factorisation des parties communes de plusieurs systèmes existants, mais qui laissent la possibilité de créer de nouveaux développements.

Le niveau système est celui considéré classiquement. Le processus de définition de l'architecture système débute à partir d'une part des besoins utilisateur concernant le système et d'autre part de l'architecture de référence issue du processus précédent du niveau domaine fonctionnel. Les besoins fonctionnels et non fonctionnels sont précisés. Pour les attributs de qualité, il est possible d'exiger des valeurs précises. Les contraintes techniques concernant l'implantation sont prises en compte.

7.4. Développement d'une architecture de référence d'un domaine fonctionnel

7.4.1. *Un exemple de domaine fonctionnel*

Afin d'illustrer notre démarche, nous considérons ici un exemple de systèmes d'information de santé dont la finalité est de gérer des dossiers médicaux partagés en support au travail de praticiens [LLO03]. A cette finalité est directement associé un ensemble de fonctionnalités au bénéfice de différents utilisateurs : les praticiens médicaux, un administrateur du système, et pour certains systèmes particuliers, des

patients eux-mêmes. De fait, nous considérons plusieurs systèmes ayant en commun un noyau de fonctionnalités très proches, presque identiques, mais se distinguant essentiellement par les exigences non fonctionnelles attendues, sensiblement différentes d'un système à l'autre, complémentées si besoin par un certain nombre d'exigences fonctionnelles additionnelles. Nous considérons dans ce domaine fonctionnel deux exemples de systèmes :

– **Dopamine**, un dossier médical informatisé à l'échelle d'une région (avec un très grand nombre de dossiers à gérer, un accès par Internet, un fort besoin de disponibilité,…) ;

– **Samarkand**, un logiciel destiné aux cabinets médicaux (un plus petit nombre de dossiers, une localisation dans un bâtiment donné, un partage de dossiers entre plusieurs praticiens associés,…).

7.4.2. *Le raffinement Fonctionnel*

Dans un premier temps, on explicite l'ensemble des fonctionnalités recensées pour l'ensemble des systèmes du domaine considéré. En mettant en évidence les éléments communs, c'est à dire en prenant l'intersection des ensembles de fonctionnalités de chacun des systèmes, on obtient un ensemble de fonctions qui constitue le noyau commun des fonctionnalités du Domaine Fonctionnel. Ce noyau commun constitue un invariant fonctionnel propre au domaine.

Dans l'exemple considéré, le noyau fonctionnel va comprendre la possibilité pour un praticien médical de se connecter au système, de rechercher la référence d'un dossier patient à partir de différents critères, de consulter ce dossier, si les droits dont il dispose le lui permettent, et d'ajouter de nouveaux items médicaux au dossier (symptômes, diagnostic, prescription, actes réalisés). Un médecin particulier ayant le statut d'administrateur médical devra pouvoir déclarer un nouveau praticien utilisateur, ou au contraire supprimer un praticien déjà déclaré. Par ailleurs, on juge que pour garantir l'évolutivité, et ce pour tout système du domaine, il est indispensable de pouvoir enrichir des référentiels médicaux (référentiels des symptômes, des diagnostics, des prescriptions et des actes), d'où la présence nécessaire d'une fonction d'enrichissement de ces référentiels, dont l'usager est le médecin administrateur. Enfin, le noyau fonctionnel doit également comprendre la capacité d'effectuer des archivages de données, sous le contrôle d'un opérateur.

Le tableau 7.4 synthétise les principales fonctionnalités communes, structurées par classes: *nécessaires* ou *possibles*, propres à tel ou tel système particulier. Les fonctions annotées comme *implicites* prennent en compte une exigence perçue en première analyse, au niveau utilisateur, comme non fonctionnelle. C'est le cas de la fonction « Login praticien », qui correspond à la prise en compte partielle d'une exigence initiale de sécurité, comme vu dans l'exemple du tableau 7.3.

Class of functionality	Functionality	User	Comment
Authenticate	Practitioner Login	Medical Practitioner	*Implicit Function*
Access to some Patient Record & add items to it	Search for a Patient Record	Medical Practitioner	
	Consult a Patient Record		
	Add an item to a Record		
Administrate	Enrich medical referential	Medical administrator	*Implicit Function*
	Add or remove a registered medical user		
Save Data	Save Data	Operator	*Implicit Function*

Tableau 7.4. *Fonctionnalités de niveau domaine*

7.4.3. *Élaboration du Modèle de Qualité Fonctionnelle*

Le Modèle de Qualité Fonctionnelle synthétise l'ensemble des exigences de qualités afférant aux fonctionnalités du domaine. Ce modèle précise, pour chaque fonctionnalité identifiée au niveau du domaine, le ou les exigences non fonctionnelles attendues modulant celle-ci (telles que performance, sécurité, …). Les exigences de niveau domaine peuvent être, pour certaines, définies de manière parfaitement précise dès le niveau domaine et s'appliquer à tout système du domaine, parce qu'intimement liées aux pratiques professionnelles considérées : ce sont les **exigences spécifiques** du niveau domaine. D'autres sont définies de manière générique et nécessitent de la définition de la valeur d'un paramètre propre à un système particulier. C'est le cas où une certaine exigence doit être présente dans tout système du domaine, avec cependant une certaine variation concernant soit le niveau de l'exigence, soit ses modalités précises. Ce sont les **exigences génériques** (telles que des exigences de disponibilité). Dans notre exemple, le modèle de qualité Fonctionnelle (FQM) est synthétisé dans le tableau 7.5.

7.4.4. *Définition de l'Architecture préliminaire*

Cette activité a pour objectif de définir une première architecture, laquelle ne tient pas encore compte, des exigences de qualité s'appliquant au domaine, et dite architecture préliminaire. Elle se traduit par le choix d'un style architectural et l'identification des composants de base, avec traçabilité fonction vers composants. Cette activité peut être commencée en parallèle de l'activité de raffinement fonctionnel, dans la mesure où le style architectural choisi ne dépend pas de manière directe du détail des fonctionnalités attendues.

Function	Characteristic / Sub Characteristic	Measured Quality Attribute [parameter]	Parameter range
Practitioner Login	**Security / Authenticity**	Authentication of medical practitioners *[Authentication method]*	-
Search Patient Record	**Performance efficiency / Time behaviour**	Maximum delay to retrieve a Patient Record ID from a given set of criteria [*max delay value*]	Max delay value ≤ 5 s
	Performance efficiency / Resources utilisation	Capacity of Patient Record store [*Min number of managed Records]*	Min number ≥ 10000)
Read Patient Record	**Performance efficiency / Time behaviour**	Delay of access to a Patient Record from its ID [*max delay value*]	Max delay value ≤ 5 s
	Security / Confidentiality	Practitioner access restricted to certain Records or Records items according to rights [*access policy*]	-
Add new item to Referential	**Maintainability / Modifiability**	Possibility to modify referential (in case of Health Standards evolution)	-

Tableau 7.5. *Modèle de Qualité Fonctionnelle de niveau domaine*

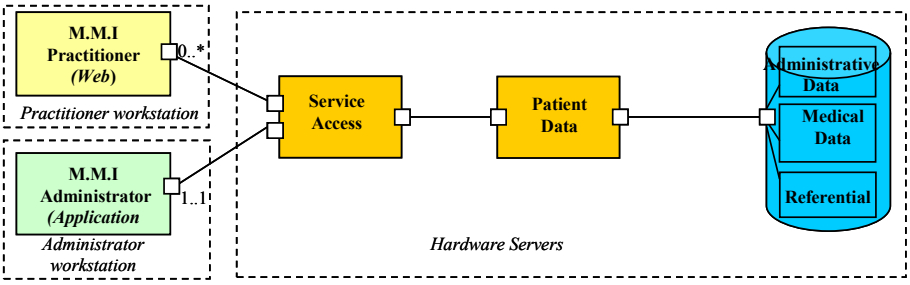

Figure 7.4. *Architecture préliminaire*

Dans notre exemple, le choix se porte sur une architecture 3-tiers comme symbolisée sur la figure 7.4. La seule contrainte de conception présente ici est de dissocier spatialement un serveur de données des postes de travail clients, lesquels sont banalisés. Les composants logiciels sont :

– dans le tiers client, les composants **M.M.I Practitioner** et **M.M.I Administrator**, respectivement interfaces Homme-Machine d'un praticien et du médecin administrateur médical ;

– dans le tiers applicatif, le composant **Service Access**, qui traite les différents types de requêtes issues des Interfaces Homme-Machine, et le composant **Medical Data** en charge des entités métiers contenues dans un dossier;

– dans le tiers de persistance, les **bases de données administrative** (données administratives des médecins et des patients), **médicale** (données médicales des patients) et la **base des référentiels** (référentiel des symptômes, diagnostics, prescription et actes).

4.5. *Élaboration du Modèle de Qualité Architecturale*

Dans le cadre de notre exemple, le modèle de qualité architectural (AQM) du domaine fonctionnel est décrit par le tableau 7.6. Pour un système à construire, chaque élément du tableau 7.6 détermine une exigence de qualité, soit directement, soit par le biais d'une instanciation spécifique via la valeur d'un paramètre.

Characteristic / Sub Characteristic	**Measured Quality Attribute [parameter]**	**Parameter range**
Security / Confidentiality	Confidentiality of stored data	-
Reliability / Availability	Guarantee of a required availability [ratio]	[0..1]
Reliability / Recoverability	Guarantee of an RTO [time] [4]	[0..10]
	Guarantee of an RPO [time] [5]	[0..10]
Usability/ User interface aesthetic	Conformance to an ergonomic chart	-

Tableau 7.6. *Modèle de Qualité Architecturale (AQM) de niveau domaine*

7.4.6. *Intégration de l'Architecture de Référence du domaine.*

Cette activité vise à la définition de l'architecture satisfaisant l'ensemble des exigences au niveau domaine. Elle sera obtenue par adaptations successives, au besoin itérativement de l'architecture préliminaire. La démarche consiste à :

– Prendre pour base l'architecture préliminaire déjà construite ;

– Intégrer les différentes exigences de qualité i) en recherchant à chaque étape différentes adaptations possibles permettant de prendre en compte l'exigence considérée, ii) en en choisissant une et en modifiant conséquemment l'architecture, iii) en vérifiant que l'architecture satisfasse toujours les exigences de qualité considérées aux étapes précédentes, iv) éventuellement en revenant en arrière remettant en cause un choix effectué à une étape antérieure.

---

[4] RTO : Recovery Time Objective

[5] RPO : Recovery Point Objective

Source (FQM | AQM)	**Function** (if from FQM)	**Characteristic / Sub Characteristic**	**Measured Quality Attribute [parameter]**	**Param range**
FQM	Practitioner Login	**Security / Authenticity**	Authentication of practitioners	-
FQM	Read Patient Record	**Security / Confidentiality**	Practitioner access restricted to certain Records or Records items according to rights	-
AQM	-	**Security / Confidentiality**	Confidentiality of stored data	-
AQM	-	**Reliability / Availability**	Guarantee of a required availability [*ratio*]	[0..1]
AQM	-	**Reliability / Recoverability**	Guarantee of an RTO [*time*]	[0..10]
AQM	-	**Reliability / Recoverability**	Guarantee of an RPO [*time*]	[0..10]
FQM	Search Patient Record	**Performance efficiency / Time behaviour**	Maximum delay to retrieve a Patient Record ID from a given set of criteria [*value*]	value ≤ 5 s
FQM	Read Patient Record	**Performance efficiency / Time behaviour**	Maximum access delay to a Patient Record from his ID [*value]*	value ≤ 5 s
FQM	Search Patient Record	**Performance efficiency / Resources utilisation**	Minimum capacity of Patient Record store [*number of managed Records]*	Min number ≥ 10000)
AQM	-	**Usability / User interface aesthetic**	Conformance to an ergonomic chart	-
FQM	Add new item to Referential	**Maintainability / Modifiability**	Possibility to modify referential (e.g. in case of Health Standards evolution)	-

Tableau 7.7: *Modèle de Qualité global de niveau domaine*

Le Modèle de Qualité Global est obtenu en fusionnant les exigences issues des modèles FQM et AQM, **ordonnées** par l'architecte. Le tableau 7.7 en montre une synthèse ordonnée par ordre de priorité, ordre basé sur l'expérience. **L'Adaptation architecturale** est réalisée par la prise en compte successive des exigences. A chaque étape, les moyens de prise en compte de l'exigence considérée sont identifiés ainsi que l'impact sur l'architecture : **immédiat**, qui doit s'imposer à tout système du domaine, ou bien **différé** avec des alternatives ou des éléments à préciser au niveau de chaque futur système à réaliser. Notre démarche est illustrée sur les premiers éléments de ce tableau, puis nous présentons une synthèse des résultats.

L'exigence « Authentification des praticiens médicaux » reste générique, différents modes d'identification étant possibles. Chaque système devra préciser son mode d'authentification requis. Les choix à opérer étant indépendants du mode d'authentification, les mécanismes spécifiques d'authentification sont encapsulés dans le composant « Authentification » ajouté. Les éléments de cette démarche sont synthétisés dans les tableaux 7.8 et 7.9. On vérifie alors bien que l'impact sur l'architecture n'implique pas de remise en cause de la propriété précédente.

Requirement nb	**1**
Characteristic / Sub Characteristic	**Security / Authenticity**
Quality requirement	Practitioner access subject to authentication
Way to take it into account / design decision	**Functional level** : possible choices – login and password recorded in the administrative database – Authentication by professional card and password – Other user authentication (biometric, etc.). **Decision**: adding "*Authentication*" component, postponing the authentication mode choice at the system level
Impact on architecture	**Immediate:** – **Adding "Authentication" component,** with allocation of a functional requirement "*Authenticate*" [mode] – **Adding requirement** to the "*Administrative Database*": add fields "office number" and "encrypted password" **Deferred**: – **Adding functional requirement** to the "Authentication" Component**:** Implement a specified authentication method – **Adding requirement on infrastructure:** Add or not a card reader and associated driver software

Tableau 7.8. *Impact architectural de l'exigence d'authentification*

Requirement nb	2
Characteristic / Sub Characteristic	**Security / Confidentiality**
Quality requirement	Restricting access to patient records according to rights
Way to take it into account / design decision	**Functional level** : possible choices – Access policy based on rights given by the doctor who created patient record (context of a doctor office) – Access policy based on practitioners role on patients record (context of a health network) **Architectural level** : possible choices – For each request on some patient medical record

	– When accessing and context management of user sessions (involves managing contexts and associated time-out)
Impact on architecture	**Immediate :** – Adding "Rights Management" component with functional requirement "Manage rights" [policy] on this component – Adding functional requirement to "Administrative Database" Component adding features "Rights of a practitioner on a record" **Deferred**: – Adding functional requirement on Component "Rights Management": Implement the specified policy at System level

Tableau 7.9. *Impact architectural de l'exigence de restriction des accès.*

L'architecture de référence du domaine fonctionnel présentée figure 7.5 est obtenue après prise en compte de l'ensemble des exigences. Les principales exigences appliquées aux composants (choix immédiats) sont indiquées.

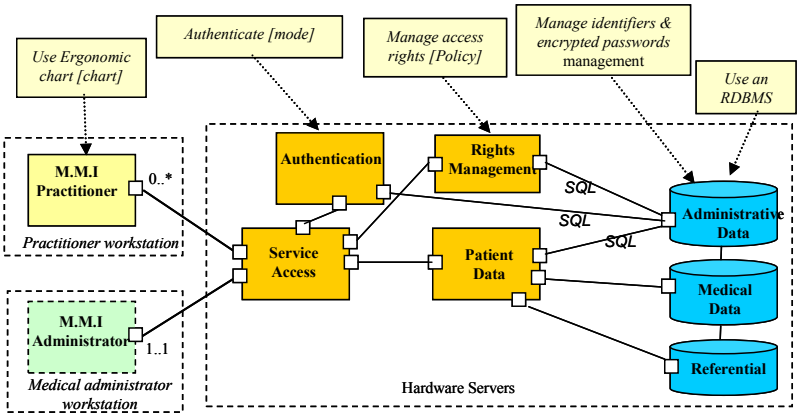

Figure 7.5. *Architecture de référence du domaine fonctionnel*

7.5. Architectures de niveau Système

Sur la base du domaine fonctionnel analysé ci-avant, nous allons à présent étudier la définition de l'architecture de deux systèmes issus de ce même domaine. Il s'agit de Dopamine, système d'information de santé régional, et Samarkand, système d'informations médicales destiné à un cabinet de praticiens. Les activités qui vont être décrites sont bien sûr spécifiques de chacun des deux systèmes considérés, et donc menées de manière absolument indépendantes. Pour la clarté de

la présentation, nous faisons le choix ici de présenter en même temps, et de manière comparative, ces deux analyses, activité par activité.

7.5.1. *Le raffinement fonctionnel*

Class of functionality	Functionality	User	Apply to
Authenticate	Practitioner Login	Medical Practitioner	Domain
	Patient Login	Medical Practitioner	**Dopamine**
Perform access to a Patient Record and add items to it	Search for a Patient Record	Medical Practitioner	Domain
	Consult a Patient Record		Domain
	Add an item to a Record		Domain
	Read his/her own (simplified) Record	**Patient**	**Dopamine**
	Grant peer access rights to a Record	**Medical Practitioner**	**Samarkand**
Administrate	Enrich medical referential	**Medical administrator**	Domain
	Add or remove a user		Domain
	Audit system accesses		**Dopamine**
Save Data	Save Data	Operator	Domain

Tableau 7.10. *Fonctionnalités de niveau domaine pour des systèmes Dopamine et Samarkand*

Pour chaque système, le tableau des fonctions communes du domaine est complété avec les fonctions spécifiques du système considéré. Nous présentons dans le tableau 7.10 l'ensemble des résultats obtenus pour chacun des deux systèmes, la dernière colonne indiquant si la fonction identifiée est requise dans l'un ou l'autre système, ou dans les deux (Domain). Les termes présents en gras indiquent les nouvelles fonctions système spécifiques.

Dans Dopamine, un accès est fourni au patient afin qu'il puisse consulter une version simplifiée de son dossier. Notons que les partages de dossiers patients entre praticiens sont régis par des politiques distinctes dans les deux systèmes : accordés explicitement par le médecin créateur du dossier dans Samarkand, mais gérés automatiquement par le système sur la base de rôles dans Dopamine.

7.5.2. Le Modèle de Qualité Fonctionnelle

Function	Characteristic / Sub Characteristic	Measured Quality Attribute [parameter]	Required value
Practitioner Login	**Security / Authenticity**	Authentication of medical practitioners [*Authentication method*]	D: **Login & password** S: **Card**
Patient Login	**Security / Authenticity**	**Authentication of patients [*Authentication method*]**	D: **Login & password**
Search Patient Record	**Performance efficiency / Time behaviour**	Maximum delay to retrieve a Patient Record ID from some criteria [*max delay*]	D: **5 s** S: **3 s**
	Resources utilisation	Capacity of Patient Record store [*Min number of managed Records*]	D: **1.000.000** S: **10.000**
Read Patient Record (practitioner)	**Performance efficiency / Time behaviour**	Delay of access to a Patient Record from its ID [*max delay*]	D: **4 s** S: **3 s**
	Security / Confidentiality	Practitioner access restricted to certain Records or Records items according to rights [*access policy*]	D: **Grants based policy** S: **Roles based Policy**
Read his own simplified Record (Patient)	**Performance efficiency / Time behaviour**	Delay of access to a Patient Record [*max delay*]	D: **5 s**

Tableau 7.11. *Modèle de Qualité Fonctionnelle des systèmes Dopamine et Samarkand*

Le tableau 7.11 présente le Modèle de Qualité Fonctionnelle des deux systèmes. Il complète celui du domaine, en y ajoutant les fonctions spécifiques de chaque système ainsi que les attributs de qualités afférant à chaque fonction. La dernière colonne indique les valeurs requises spécifiques pour chacun des deux systèmes, attribut par attribut. Une exigence de conception est ajoutée de plus dans Dopamine : le système pourra être distribué par Internet avec un site serveur d'une part, et différents postes clients répartis géographiquement d'autre part.

7.5.3. *Architecture de base*

De manière tout à fait analogue au processus qui a permis de construire l'architecture de référence du domaine fonctionnel, l'architecture d'un système est

construite par adaptations successives de l'architecture de référence du domaine fonctionnel, sur la base des exigences de qualité précisées pour chaque système.

7.5.4. *Le Modèle de Qualité Architecturale*

Le tableau 7.12 présente dans un cadre unique les AQM des deux systèmes reprenant l'AQM du domaine, en y ajoutant les exigences de qualité architecturale spécifiques des systèmes considérés. Les valeurs caractéristiques des paramètres permettant d'instancier les exigences génériques sont précisées. Une des exigences de qualité architecturale ajoutée dans Dopamine concerne la sécurité des communications, ce système étant déployé sur Internet, à la différence de Samarkand, basé sur un réseau privé d'un même site géographique. Les autres exigences du tableau 7.12 sont des exigences génériques de niveau domaine, instanciées dans le contexte de chaque système. Sont omises les exigences déjà prises en compte au niveau Domaine, c'est-à-dire ne comportant aucun choix différé.

Characteristic / Sub Characteristic	**Measured Quality Attribute [parameter]**	**Required value**
Security /Confidentiality & Integrity	**Confidentiality and Integrity of communications**	Applies to: D
Reliability / Availability	Guarantee of a required availability [ration]	D: **0.999** S: **0.99**
Reliability / Recoverability	Guarantee of an RTO [time]	D: **10 mn** S: **30 mn**
	Guarantee of an RPO [time]	D: **1 mn** S: **30 mn**
Usability / User interface aesthetic	Conformance to an ergonomic chart	D: **Practitionner & Patient's charts** S: **Practitionner's chart**

Tableau 7.12. *Modèle de Qualité Architecturale des systèmes Dopamine et Samarkand*

7.5.5. *Architecture des systèmes Dopamine et Samarkand*

Pour un système donné, le FQM et l'AQM sont fusionnés dans un Modèle de Qualité global, qui reprend l'ensemble des exigences de deux modèles en les ordonnant. Comme pour le niveau du domaine, ce classement repose sur l'expertise de l'architecture, et définit un ordre de prise en compte de chaque exigence dans le processus d'adaptation de l'architecture du domaine. Par souci de simplification de la présentation, et de manière à continuer la mise en parallèle des architectures à construire, nous avons supposé que cet ordre était ici le même pour les deux systèmes.

Requirement	Analysis / Decisions	Impact
Security /Authenticity **Authentication of medical practitioners** *[Authentication method]* (Function: Practitioner Login)	**Analysis & Decision :** The *Authentication* component implements the authentication mode for the practitioner selected for the considered system	**Dopamine**: **adding** - **functional requirement:** the *Authentication* component for practitioner by Login & password **Samarkand**: **adding :** - **functional requirement:** the *Authentication* component for practitioner by card **- requirement on infrastructure**: Adding of a card reader
Security /Authenticity **Authentication of patients** *[Authentication method]* (Function: Patient Login)	**Analysis & Decision :** Dopamine The *Authentication* component implements the authentication mode for the Patient Samarkand : none	**Dopamine: adding** **- functional requirement:** the *Authentication* component for patient by Login & password **Samarkand**: -
Security / Confidentiality **Practitioner access restricted to certain Records or Records items according to rights** [*access policy*] (Function: Read Patient Record)	**Analysis & Decision :** The Rights Management component implements political rights management as required in the considered system	**Dopamine: adding :** **- functional requirement:** the *Rights Management* component implementing *Roles Policy* **Samarkand**: **adding** - **functional requirement: :** the *Rights Management* component implementing *Grants Policy*
Security / Confidentiality & Integrity **Confidentiality & integrity of communications**	**Analysis & Decision :** Dopamine : Secure protocols and PKI management Samarkand : none (private network)	**Dopamine: adding** **- requirement on infrastructure:** Implementation of secure protocol with PKI management **Samarkand**: -
Reliability / Availability **Required availability** [*required availability ratio*]	**Analysis & Decision :** Choosing a hardware server suitable to required availability Dopamine : suitable hardware server Samarkand : none (standard hardware server)	**Dopamine: adding** **- requirement on infrastructure:** Suitable hardware server **Samarkand**: *néant*

Tableau 7.13. *Extrait des décisions de conception et impacts sur l'architecture pour les systèmes Dopamine et Samarkand*

De manière tout à fait analogue à la construction de l'architecture de référence du domaine, l'architecture de chaque système est construite sur la base d'une suite de décisions d'adaptation, se traduisant chacune par des ajouts de composants et/ou des ajouts de nouvelles exigences sur des composants déjà identifiés ou encore sur l'infrastructure d'exécution. A titre d'exemple, nous présentons ci-dessous dans le tableau 7.13 un extrait des résultats obtenus pour les premières exigences traitées (les 5 premières exigences sur un total de 11), en précisant ce qui s'applique à l'un ou l'autre système.

Nous présentons les architectures des systèmes Samarkand et Dopamine : sur la figure 7.6 l'architecture à laquelle conduisent les décisions de conception afférant au système Samarkand et sur la figure 7.7 celle du système Dopamine.

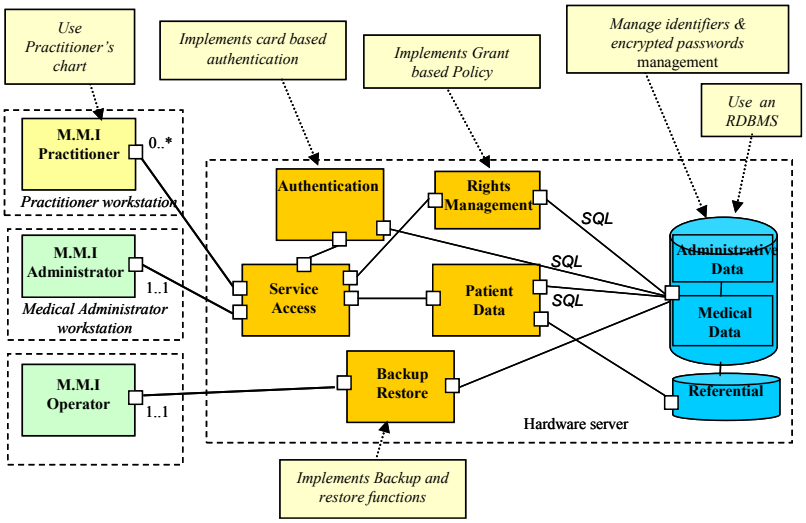

Figure 7.6. *Architecture du système Samarkand*

Ces deux exemples de systèmes issus du même *domaine fonctionnel* mettent en évidence le fait que deux ensembles d'exigences fonctionnelles très proches (ici dans notre cas l'un est un sur ensemble de l'autre), mais avec exigences de qualité différentes, conduisent à des architectures bien distinctes. Ainsi, les deux architectures présentées diffèrent-elles par des composants additionnels requis dans la seconde architecture (ici, permettant d'assurer la disponibilité requise), la topologie des connexions entre composants, mais aussi par différentes *variantes* de mêmes composants de base, lesquelles assurent une même fonction, mais sur la base de différentes exigences de qualité et différentes exigences de réalisation.

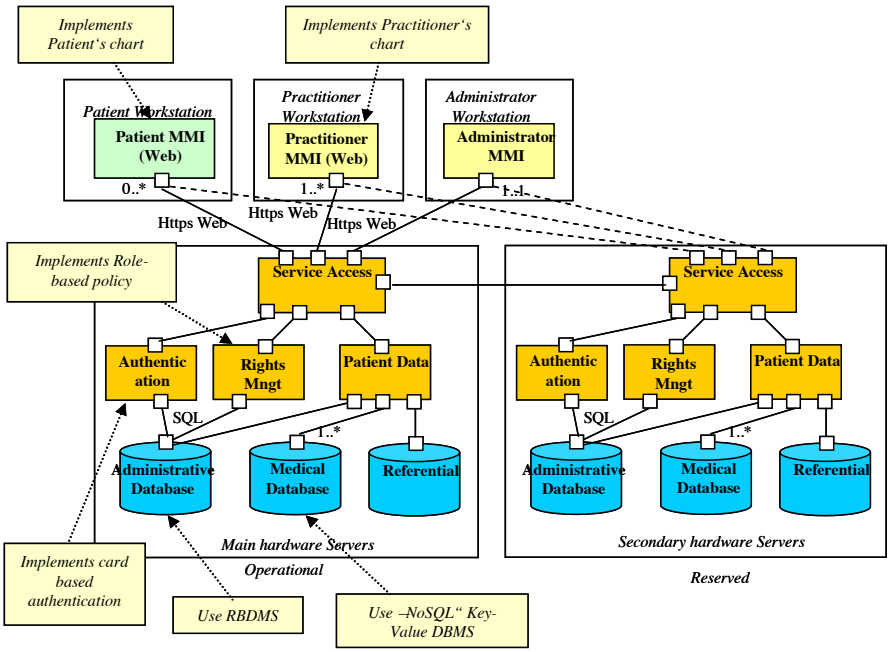

Figure 7.7. *Architecture du système Dopamine*

7.6. Travaux connexes

Dans le domaine de la conception architecturale, il est très essentiel de garder un lien entre spécifications des exigences et solution architecturale. [CAS12] mentionne les problèmes de spécifications des exigences informelles et /ou incomplètes et la complexité de maintenir les liens de traçabilité. Malgré ces défis, il existe peu d'approches s'intéressant à l'analyse et à la compréhension des exigences dans le but de proposer un ensemble de solutions architecturales y répondant de manière satisfaisante. Or les techniques et les méthodes de développement de logiciels devraient inclure une façon de traiter systématiquement la relation entre les modèles d'exigences et architecturaux.

Les exigences fonctionnelles sont prises en compte au niveau de la spécification du système et les non fonctionnelles par l'architecture. L'étape de conception architecturale consiste à choisir et à appliquer une solution qui réponde au mieux

aux exigences de qualité. En conséquence, toutes les méthodes de conception architecturale doivent traiter les exigences de qualité et aussi, inclure un processus d'évaluation de la qualité résultante. Plusieurs méthodes traditionnelles d'évaluation d'architecture sont présentées dans [RAS11], toutes basées sur ATAM (Attribute Tradeoff Analysis Method) [BAS02a] que nous allons commenter, et une version adaptée de l'approche famille de produits, FAAM (Family Architecture Analysis Method) [DOL02].

ATAM (Attribute Tradeoff Analysis Method) [BAS02a] est une méthode d'évaluation architecturale bien connue proposée par le SEI[6]. Le processus ATAM consiste à réunir les parties prenantes pour analyser les besoins (faire un brainstorming) et déterminer les attributs de qualité (on parle de caractéristiques de qualité dans la terminologie de l'ISO / IEC 25010) qui seront utilisés pour définir des scénarios. Ceux-ci seront associés aux attributs de qualité par le biais d'un arbre d'usage (utility tree), ainsi qu'aux contraintes et mesures à prendre en compte. Les scénarios sont triés par priorité pour chaque attribut de qualité et utilisés en conjonction avec les solutions architecturales et une étude des risques pour permettre de faire des arbitrages [BAR98]. ATAM utilise les structures ABAS (Attribute - Based Architectural Analysis), qui est un cadre pour définir des solutions architecturales ainsi que leurs attributs de qualité; ATAM n'utilise pas un modèle de qualité, mais l'arbre d'usage est conceptuellement semblable. Un modèle de qualité standard peut être utilisé dans ABAS pour spécifier les exigences de qualité de la solution architecturale [LOS03]. L'analyse ATAM est qualitative, s'appuyant sur l'expérience de l'architecte et de l'équipe de travail.

Les approches guidées par les scénarios permettent de définir le modèle de qualité d'un système logiciel [TAN12]. Celles-ci représentent une manière de mettre en relation les exigences fonctionnelles et non fonctionnelles avec leurs besoins en terme de qualité. Les méthodes de conception architecturales classiques, telles que ADD [BAS02][WOJ06], considèrent les exigences de qualité définies à l'aide de scénarios basés sur la structure de l'architecture. Par exemple, la sécurité, la fiabilité ou la performance seront décrites en terme de résultats liés à une exécution particulière visée et décrite sur la base d'interactions entre les composants de l'architecture. Ces scénarios contraignent la conception de l'architecture [KOT98]. Cependant, ces méthodes de conception ne permettent pas de considérer le fait que certaines fonctionnalités requièrent également des qualités qui sont comme des fonctionnalités implicites, résolues ou prises en charge par des mécanismes particuliers, solutions architecturales ou patterns. Il est important de considérer et de mémoriser ces mécanismes ou composants implémentant ou prenant en charge une exigence de qualité qui est une fonctionnalité implicite. Des exemples de tels

[6] Software Engineering Institute, Carnegie Mellon University

composants sont les proxys ou les « *event channel* » qui sont utilisés pour garantir la sécurité (intégrité, authenticité et confidentialité) d'une communication. Les approches ligne de produits avec leur modèle de caractéristiques (« *feature models* ») contenant les points de variabilité permettent de mémoriser les choix qui peuvent être faits de tel ou tel composant.

Parmi les méthodes classiques de conception d'architecture on peut distinguer deux grandes catégories :

– Les méthodes orientées fonctionnalité telles que I. Bosch [BOS99] et RUP [KRU99] qui commencent par définir les fonctionnalité requises puis transforment l'architecture initiale ainsi définie pour répondre aux exigences de qualité du système, par ajout / suppression de composants;

– Les méthodes orientées qualité :

- les approches orientées but de A. van Lamswerdee [LAM03], Chung et al. [CHU03]. Ces méthodes identifient des objectifs abstraits principaux du système ("hardgoals" et "softgoals") représentant l'intention d'un utilisateur ou une fonctionnalité. Les exigences non fonctionnelles ("softgoals») ou de qualité, sont généralement des exigences architecturales de haut-niveau du système, qui sont raffinées jusqu'à trouver des solutions architecturales. Les fonctionnalités du système sont alors associées à ces solutions architecturales.

- les approches incrémentales et itératives de Weiss [ARA02] et ADD [BAS02b] sont basées sur des scenarios portant sur la structure de l'architecture ; à chaque itération, les fonctionnalités sont introduites comme instanciation de composants et connecteurs.

Microsoft [MIC09] recommande une méthode itérative et incrémentale de conception d'architecture logicielle caractérisée par :

- l'utilisation d'objectifs architecturaux clairs et de scénarios clés,

- l'identification du type d'applications, styles et techniques (à rapprocher de notre description de domaine),

- l'identification de questions clé concernant les attributs de la qualité et des préoccupations transverses,

- la définition d'une première solution candidate à partir de laquelle il est possible ensuite d'itérer pour évaluer la solution courante vis à vis des principaux scénarios.

Concernant les scénarios, ils sont considérés comme très importants pour le succès de l'application. Ils peuvent être définis comme n'importe quel scénario répondant à un ou plusieurs des critères suivants : il représente un enjeu, une problématique inconnue importante ou une problématique potentiellement risquée, il se réfère à un cas d'utilisation architectural important, il représente la synthèse

d'attributs de qualité et de fonctionnalités, il représente un compromis entre différents attributs de qualité. Ce processus architectural se veut une approche itérative et incrémentale. La première architecture candidate sera une conception de haut niveau sur laquelle les scénarios clés peuvent être évalués, vérifier que les exigences et les contraintes connues sont bien prises en compte, les attributs de qualité quantifiés grossièrement et le cadre architectural défini.

Une approche très prisée des industriels est l'approche lignes de produits. La grande différence avec notre approche est qu'un domaine fonctionnel est ouvert à de nouvelles fonctionnalités ou systèmes, alors que dans une ligne de produits, toutes les fonctionnalités sont prévues et décrites. Les décisions à prendre sont connues dès le départ et l'accent est mis sur la variabilité là où elle doit être prise en compte. Notre démarche est plutôt focalisée sur la prise en compte systématique des propriétés de qualité qui sont nécessaires et déterminantes à la construction d'une architecture de base. Celle-ci est peut-être vue comme un élément réutilisable (*asset*) ou une architecture de référence ou encore l'architecture de la ligne de produits. Elle exprime une partie importante de la connaissance du domaine. Notons que le raffinement des caractéristiques de qualité de l'ISO 25010 permet de prendre en compte la variabilité non fonctionnelle. Ainsi, pour chaque sous caractéristique, plusieurs solutions sont présentées qui peuvent être implémentées ou mises en œuvre dans des produits différents. Cet aspect non fonctionnel n'est pas très bien traité dans les approches SPL, où il est en général laissé de coté ou non explicitement pris en compte [BEN05], [SIE08], [BAR10], [TAN12].

Le travail de [HOF07] compare cinq méthodes de conception architecturale traditionnelles basées sur des scenarios et utilisées par les industriels pour le développement d'architectures de familles de produits : la méthode ADD (« Attribute-Driven Design » [BAS02b]) ; la méthode 4 vues de Siemens (SV4) [HOF99], la méthode 4 + 1 vues de RUP [KRU99], la méthode BAPO (« Business Architecture Process and Organization ») de Philips Research en considérant les 5 vues du modèle CAFCR « Customer, Application, Functional, Conceptual and Realization ») [AME03], et ASC (« Architectural Separation of Concerns ») [RAN00] de Nokia. A l'issue de la comparaison, les auteurs proposent un modèle général de conception centré architecture incluant des activités d'évaluation et d'analyse qui peut être utilisé soit pour une famille de produits, soit pour un seul système, comme dans notre cas. Nous allons détailler cette méthode :

Activités de conception non séquentielles :

– L'analyse architecturale permet de définir le problème à résoudre par l'architecture ;

– La synthèse architecturale propose des solutions architecturales à un ensemble significatif d'exigences architecturales, permettant de passer de l'espace du problème à l'espace de la solution ;

– L'évaluation architecturale s'assure que les décisions de conception architecturale sont correctes, les solutions candidates sont évaluées en fonction des exigences architecturales importantes.

Documents produits au cours des activités :

– Les contraintes architecturales, en général, il s'agit d'exigences non fonctionnelles concernant le système et la manière dont il doit être implanté, elles peuvent inclure l'utilisation de normes ;

– Le contexte est un objectif général de l'entreprise ou une caractéristique d'organisation ;

– Les besoins spécifiques concernant l'architecture peuvent dériver d'autres préoccupations architecturales ou du contexte du système ;

– La solution architecturale candidate présente des solutions alternatives ou partielles; elles comprennent des informations concernant la logique de conception (les décisions prises, leur traçabilité depuis les exigences, etc.).

Certaines données pour la conception architecturale sont moins explicites :

– La connaissance concernant la conception : il s'agit de la connaissance de l'architecte, de la mémoire de l'entreprise ou de la communauté ;

– La connaissance de l'analyse : les patterns d'analyse, les modèles analytiques, les méthodes et les techniques de workflow ;

– La connaissance concernant la réalisation : la technologie, les composants, la gestion de projet.

En général ces connaissances permettent d'évaluer la conception pour s'assurer que le système peut être construit ou la qualité du résultat. Cette méthode synthétise les savoir-faire actuels des architectes. Remarquons cependant que d'une part elle décrit, tout comme les méthodes dont elle est dérivée, des processus pour la conception architecturale de nouveaux systèmes, et non comme notre approche, par réutilisation de connaissances et en partie de systèmes existants; et d'autre part qu'il n'y est pas fait mention de modèles de qualité, le notre étant de plus basé sur le standard ISO/IEC 25010.

7.7. Conclusion

Le développement d'un système est un travail complexe et de multiples approches ont été décrites pour permettre de systématiser les étapes et de garantir une meilleure adéquation du résultat vis à vis des besoins de l'utilisateur. Dans cet

article, nous nous sommes intéressés à l'étape de développement de l'architecture du futur système, étape prenant en compte les exigences non fonctionnelles et de qualité.

Dans la pratique, il est bien rare de développer un système à partir de rien. Cependant, les méthodes usuelles ont du mal à décrire comment capitaliser les connaissances d'une équipe de projet. Les méthodes décrivent soit un nouveau travail, soit des évolutions d'un système, soit dans le cadre des lignes de produit, toutes les variantes possibles d'un produit par réutilisation de composants existants. Notre approche est différente en ce qu'elle propose de capitaliser les connaissances dans ce que avons appelé un domaine fonctionnel. Des points de variabilité peuvent être introduits s'ils sont connus, mais ce n'est pas la partie importante. Les connaissances d'un domaine fonctionnel sont celles partagées par tous les systèmes d'un certain domaine, c'est un noyau de connaissance qui permet de définir une architecture de référence. Les différents systèmes seront développés à partir de cette connaissance capitalisée et de l'architecture partagée.

Dans cet article, nous avons donc proposé une méthode de développement d'architecture de systèmes en les replaçant dans le cadre de familles de systèmes de fonctionnalités proches. Notre méthode s'applique aussi bien pour définir l'architecture de référence d'un domaine fonctionnel que celle d'un système particulier dérivé. Notre démarche décrit, comme dans un cycle en Y, les travaux menés en parallèle d'une part par l'analyste étudiant les propriétés fonctionnelles et d'autre part par l'architecte travaillant sur les aspects techniques. Leurs travaux sont synthétisés en fin de démarche dans une architecture dans laquelle les fonctionnalités sont assignées à des composants ainsi que les propriétés non fonctionnelles.

Ainsi pour définir un système, la première étape consiste à définir son domaine fonctionnel, puis à en dériver son architecture spécifique.

Nous avons appliqué et détaillé les deux étapes de notre méthode sur une étude de cas issue du monde professionnel de la santé, le développement des systèmes Dopamine et Samarkand de partage de fiches patient. Ces deux systèmes existent et originellement avaient été développés de manière ad hoc, sans démarche précise. Mais la capitalisation des connaissances montre qu'en l'occurrence, ces deux systèmes partagent beaucoup de choses et auraient pu être développés comme montré ici. Nous sommes sûrs que notre approche s'applique à de nombreux cas, où les systèmes différent mais l'architecture et les grandes fonctionnalités se ressemblent.

Pour aller plus loin dans l'aide au développement, il est nécessaire de formaliser l'approche. Pour cela, il faut d'une part pouvoir gérer les liens de traçabilité entre les

différents niveaux d'abstraction, langages plus ou moins formels et systèmes pas forcément partagés. Les méthodes actuelles ne permettent pas un tel niveau de flexibilité pour gérer les liens de traçabilité entre les besoins et les composants architecturaux. D'autre part, la formalisation de notre approche permettra de décrire des outils d'aide à l'évaluation de la prise en compte au niveau architectural des propriétés non fonctionnelles et de guider les architectes et analystes. En ce qui concerne la définition de l'architecture de référence à partir de systèmes existants proches fonctionnellement, un travail a déjà commencé [LOS12] qui pourra prendre appui sur ce présent travail.

Bibliographie

[AME03] P. America, H. Obbink, E. Rommes, *Multi-view variation modeling for scenario analysis*, In proceedings of fifth Workshop on Product Family Engineering (PFE-5), Siena, Italy, Springer Verlag, 44-55, 2003.

[ARA02] J. Araujo, M. Weiss, *Using the NFR framework to represent patterns*, PLoP-02, 2002.

[BAR98] M. Barbacci, P. Feiler, M. Klein, H. Lipson, T. Longstaff, C. Weinstock, and S. Carriere, *Steps in an Architecture Tradeoff Analysis Method: Quality Attribute Models and Analysis* (CMU/SEI-97-TR-029), lu en mars 2013, SEI, Carnegie Mellon University, 1998.

[BAS02a] L. Bass, M. Klein, F. Bachmann, *Quality Attribute Design Primitives and the Attribute Driven Design Method*, Software Product Family Engineering Int. Workshop PFE 2001, Bilbao, Spain LNCS 2290, 2002, 169-186.

[BAS02b] L. Bass, M. Klein, F. Bachmann, *Attribute Driven Design Method (ADD)*, SEI, CMU, 2002.

[BAR10] J. Bartholdt, R. Oberhauser, M. Medak, and A. Rytina, *Integrating Quality Modeling in Software Product Lines*, International Journal on Advances in Software, vol 3 no 1 & A, 161-174, 2010.

[BEN05] D. Benavides, P. Trinidad, A. Ruiz-Cortés Automated Reasoning on Feature Models (2005), LNCSm Advanced Information Systems Conference: 17th International Conference, CAISE 2005

[BOS99] J. Bosch, *Design and Use of Software Architecture*, Addison Wesley, Harlow, England, 2000.

[BUS96] F. Buschmann, R. Meunier, H. Rhonert, P. Sommerlad, M. Stal, P*attern-Oriented Software Architecture, A System of Patterns*, Wiley & Sons, New York, 1996.

[CAL97] J. McCall, P. Richards and D. Walters, *Factors in Software Quality*, Rome Aide Defense Center, Italy, 1997.

[CAS12] J. Castro, M. Lucena, C. Silva, F. Alencar, E. Santos, J. Pimentel, *Changing attitudes towards the generation of* architectural *models*, Journal of Systems and Software (JSS), Volume 85, Issue 3, 463–479, March 2012.

[CHU03] L. Chung, K. Cooper, A. Yi, Developing adaptable software architectures using design patterns: an NFR approach, Computer Standards & Interfaces 25 (2003) 253-260.

[CRO84] P.B., Crosby, *Quality without tears*, McGrow-Hill Books, New York, USA, 1984.

[DEM00] W. E. Deming, *Out of the Crisis*, MIT press, Cambrige, Mass, 168-169, 2000.

[DOL02] T. Dolan, *Architecture Assessment of Information-System Families*, Unpublished Ph.D. Thesis, University of Technology, Eindhoven, The Netherlands, 2002.

[HOF07] C. Hofmeister, P. Krutchen, R. L. Nord, H. Obbink, A. Ran, P. America, *A general Model* of *Software Architecture design derived from five industrial approaches*, Journal of Systems and Software 80, 106-126, 2007.

[HOF99] C. Hofmeister, R. Nord, D. Soni, *Applied Software Architecture*, Addison-Wesley, Boston, 1999.

[IEE83] IEEE *Std 729*-1983.

[ISO06] FDIS 25030: *Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - Quality Requirements,* ISO/IECJTC1/SC7/N3632, 2006.

[ISO11] ISO/IEC 25010: *Systems and software* engineering - *Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models*, ISO/IEC JTC1/SC7/WG6, 2011.

[JEA12] S. Jean, F. Losavio, A. Matteo, N. Levy, *Standards de qualité et préférences utilisateurs pour la modélisation des propriétés non fonctionnelles dans OWL-S*, Techniques et science informatique (TSI), Vol 21, No. 1, 39-70, 2012.

[KOT98] G. Kotonya and I. Sommerville, *Requirements engineering, processes and techniques*, Wiley ed. 1998.

[KRU99] P. Krutchen, *The Rational Unified Process an Introduction,* Addison-Wesley-Longman, Reading, MA, 1999.

[LAM03] A. Van Lamsweerde, *From system goals to software architecture*, In Formal Methods for Software Architectures, M. Bernardo & P. Inverardi (eds.), LNCS, Springer-Verlag, 2003.

[LLO03] D. Lloyd and D. Kalra, *EHR Requirements*, Centre for Health Informatics and Multi-professional Education, University College, London, 2003, http://discovery.ucl.ac.uk/1583/1/A5.pdf

[LOS03] F. Losavio, L. Chirinos, N. Lévy, A. Ramdane-Cherif, *Quality Characteristics for Software* Architecture. J. of Object Technology, Vol 2, No. 2, 133-150, March/April 2003.

[LOS09] F. Losavio, A. Matteo, N. Lévy, *Web Services Domain Knowledge with an Ontology on Software Quality Standards*, ITA'09, 74-85, Glyndwr, UK, September 2009.

[LOS12] F. Losavio, O. Ordaz, N. Lévy, A. Baiotto, *Refactoring Process for Product Line Architecture Design*, Journée Lignes de Produits (JDLP) 2012, 47-58, Lille, France, Nov. 2012. http://jldp.org/2012/images/jldp2012-actes.pdf

[LOS13] F. Losavio, A. Matteo. *Reference Architecture Design Using Domain Quality View*, Journal of Software Engineering and Methodology, Vol 3, No. 1, 47-61, March 2013.

[MIC09] Microsoft, *Microsoft Application Architecture Guide – Software Architecture and design*, 2nd. Edition, chap. 4, 2009, http://msdn.microsoft.com/en-us/library/ee658084.aspx

[ROC01] P. Rocques, F. Vallée, *UML en action,* Eyrolles, 2e édition, mars 2001.

[POH05] K. Pohl, G. Böckle, F. van der Linden. *Software product line engineering - foundations, principles, and techniques,* Springer IXXVI, 1-467, 2005.

[RAN00] A. Ran, *ARES Conceptual Framework for Software Architecture*. In M. Jazayeri, A. Ran, F. van der Linden (Eds.), Software Architecture for Product Families Principles and Practice, Addison- Wesley, Boston, 1-29, 2000.

[RAS11] A. Rashid, JC. Royer and A. Rummler (Eds), *Aspect-Oriented Model-Driven Software* Product *Lines*. *The AMPLE Way,* Chap. 5. Cambridge Univ Press, Cambridge, 2011.

[SIE08] N. Siegmund, M. Kuhlemann, M. Rosenmuller, C. Kastner, G. Saake. *Integrated Product Line Model for Semi-Automated Product Derivation Using Non-Functional Properties*, VaMoS, 25-32, Essen, Germany, 16-18 January 2008.

[SG96] M. Shaw, D. Garlan, "Software Architecture. Perspectives of an emerging discipline", Prentice-Hall, 1996

[TAN12] L. Tan, Y. Lin, H. Ye. *Quality-Oriented Software Product Line Architecture Design*, Journal of Software Engineering and Applications, 5, 472-476, 2012.

[WOJ06] R. Wojcik et al. *Attribute-Driven Design (ADD), Version 2.0*, Carnegie Mellon Software Engineering Institute, 2006. http://www.sei.cmu.edu/reports/06tr023.pdf.