Doter la maîtrise d'ouvrage d'une surveillance stratégique en temps réel de la disponibilité de son système d'information

Vincent Ottinger

CONSERVATOIRE NATIONAL DES ARTS ET MÉTIERS
CENTRE REGIONAL ASSOCIE DE LYON

MÉMOIRE
présenté en vue d'obtenir
le DIPLÔME d'INGÉNIEUR CNAM

SPÉCIALITÉ : INFORMATIQUE

OPTION : SYSTÈME D'INFORMATION

par

Vincent OTTINGER

Doter la maîtrise d'ouvrage d'une surveillance stratégique en temps réel de la disponibilité de son Système d'Information

Soutenu le …….

JURY

PRÉSIDENT : Christophe PICOULEAU

MEMBRES : Bertrand DAVID

Claude GENIER

Roland BURGNIARD

Didier LECLERC

Remerciements

Travaillant au Centre des Technologies de l'Information (CTI) de l'État de Genève depuis plus de 7 ans, j'ai la chance d'avoir un parcours qui m'a permis de découvrir les différents aspects constituants un système d'information. En commençant par la définition du besoin, en passant par le développement, puis en travaillant sur les aspects de mise en production, j'ai également eu un aperçu des contraintes de mise en place de systèmes hautes disponibilités. En tant que coordinateur pour le service Sécurité, je suis impliqué sur le composant le plus transversal, utilisé par tous les projets et ayant un fort impact sur l'ensemble des utilisateurs de l'État de Genève.

Je tiens à remercier en premier lieu mes collègues du service Sécurité du CTl et plus particulièrement mon responsable, Monsieur Roland Burgniard, qui me soutient dans tous mes projets depuis près de 5 ans et qui m'a encouragé lorsque j'ai décidé de reprendre mes études à l'EiCNAM il y a 3 ans, en parallèle de mon travail.

Je remercie également Monsieur Joaquin Romero pour son aide, pour m'avoir mis à disposition un environnement de prototypage qui m'a permis de réaliser mon POC et faire les tests nécessaires, et surtout pour le temps qu'il m'a consacré en dehors de ses heures de travail.

Merci à Monsieur Claude Genier, pour son aide et son encadrement sur ce mémoire.

Un merci particulier à Cécile Biensan, Cindy De Coulon, Élisabeth Ottinger, Jean-Daniel Ciana, Dominique Climenti, Jérôme Dinichert, Cédric Engeler, Nicolas Frankel, Patrick Grogniet, Patrick Jane, Didier Leclerc, Mario Matos, Antoine Ruffel et Raphaël Zanetta.

Abréviations

AEL Administration En Ligne. Ensemble des services de l'administration genevoise offerts aux citoyens et entreprises

API Application Programming Interface. Interface fournie par un programme informatique pour l'interaction avec d'autres programmes.

CMDB Configuration Management DataBase. Référentiel des composants d'un système d'information, défini dans ITIL.

CTI Centre des Technologies de l'Information. Service informatique du Canton de Genève

DNS Domain Name Server. La résolution DNS traduit une adresse de domaine en texte (ex. : www.google.ch) par son adresse IP physique (ex. : 74.125.43.147)

GINA Gouvernance des Identités Numériques et des Accès. Socle sécurité de l'État de Genève

HTTP Hyper Text Transfert Protocol. Protocole de communication du Web.

ITIL Information Technology Infrastructure Library. Ensemble d'ouvrages recensant les bonnes pratiques pour la gestion de services informatiques.

J2EE Java Enterprise Edition. Spécification de Sun (www.sun.com) pour définir une approche structurelle multiniveaux pour les applications Java.

MOA Maîtrise d'Ouvrage. Les clients d'un projet.

OLA Operational Level Agreement . Accord entre les fournisseurs de services d'un même centre informatique, qui définit le niveau d'exigences entre les services et composants informatiques.

POC Proof Of Concept. La preuve d'un concept est une réalisation courte ou incomplète d'une méthode ou d'une idée pour démontrer sa faisabilité.

RAID Redundant Array of Independent Disks. Architecture de composant avec plusieurs disques durs, afin de dupliquer l'information pour augmenter la disponibilité.

SI Système d'information

SLA Service Level Agreement. Accord entre un fournisseur de services et un client permettant de définir le niveau de service attendu.

SNMP Single Network Management Protocol. Protocole de communication qui permet la gestion, l'interrogation et la surveillance de composants informatiques à distance.

UML Unified Modeling Language. Langage de modélisation graphique à base de pictogrammes. Il permet de décrire le système informatique.

Glossaire des termes techniques

Access-point Point d'accès WiFi, permettant à un PC de se connecter au réseau sans fil.

Authentification forte Authentification avec un second facteur d'authentification.

Centre de Service Département de l'entreprise qui assure le service d'assistance informatique du processus Exploitation des Services selon ITIL.

Chaine de sauvegarde Ensemble de sauvegardes qui sont faites de manière automatique sur plusieurs serveurs.

Cluster Ensemble de serveurs, au minimum deux, garantissant une redondance entre eux pour assurer une continuité de service et/ou répartir la charge de calcul et/ou la charge réseau.

dotNET ou **.NET** Langage de programmation, permettant notamment de porter des applications au format web.

Test end-user Test effectué selon le point de vue d'un utilisateur, c'est-à-dire sur un poste de travail équivalent.

Heartbeat Signal échangé entre des composants d'un cluster pour tester la disponibilité des différents éléments du cluster, c'est un élément important pour construire des architectures de haute disponibilité.

Logiciel Open Source Logiciel dont le code source est disponible. Il existe différents types de licence open source.

MS SQL Serveur Système de gestion de bases de données de Microsoft.

MySQL Système de gestion de bases de données open source.

Nagios Application open source permettant de faire de la surveillance de

composants informatiques.

Nexthink Solution d'analyse comportementale d'un poste de travail utilisateur.

Onduleur Système de batteries de secours permettant de délivrer un courant électrique lissé en cas de surtension ou de microcoupures du réseau électrique.

Oracle Entreprise informatique spécialisée principalement dans les systèmes de gestion de bases de données.

Patrol Solution de surveillance applicative de la société BMC Software.

Perl Langage de programmation informatique interprété.

PHP Langage de programmation web libre de droits et gratuit.

Service d'astreinte Système où une personne doit être atteignable 24/24h, 7/7j pour intervenir en cas de problème.

Ping Commande informatique permettant de contrôler la communication entre deux machines d'un réseau.

Pingdom Solution de surveillance d'application depuis un site web.

Routeur Élément complexe reliant plusieurs segments d'un réseau et permettant de faire de la redirection de paquets entre les différents segments.

Serveur Apache Serveur web HTTP libre, le plus utilisé sur le web.

Switch Élément reliant plusieurs segments d'un réseau.

Ticket d'incident ou **Ticket de panne** Fiche électronique décrivant un incident et permettant d'avoir un suivi du traitement de la panne jusqu'à sa

résolution.

Version bêta Préversion d'un logiciel qu'un groupe d'utilisateurs peut tester afin d'en identifier les anomalies. Une version bêta ne doit pas être utilisée dans un environnement de production.

Webservice Connecteur ou architecture orientée services, basé sur le protocole HTTP permettant la communication et l'échange de données entre applications et systèmes hétérogènes dans des environnements distribués.

Zabbix Outil de surveillance open source.

Plan

Introduction

Le Centre des Technologies de l'Information, ci-après nommé « CTI », est le centre informatique de l'État de Genève. Il est responsable de fournir un service informatique de qualité à près de 25'000 utilisateurs.

Ce mémoire a pour sujet la mise à disposition pour la maîtrise d'ouvrage (MOA), donc pour le client, d'une surveillance de son système d'information. C'est-à-dire avoir une approche de la modélisation et de la surveillance à partir des services métiers, destiné à l'utilisateur final.

Ma spécialité est le contrôle des accès applicatifs. La surveillance d'applications ne fait pas partie de mes attributions. Néanmoins, le service Sécurité pour lequel je travaille est responsable du socle sécurité de l'État de Genève qui est un composant transversal nécessaire pour se connecter aux applications. Si une application ne fonctionne pas, l'utilisateur reste en attente sur la fenêtre d'authentification et pense généralement que c'est ce composant qui pose problème. De nombreux cas d'incidents remontent par erreur jusqu'à nous avec ce type de diagnostic. Il nous paraît essentiel d'avoir un moyen de contrôle rapide pour diagnostiquer quel composant est réellement responsable de l'indisponibilité.

La complexité engendrée par la mise à disposition pour la maîtrise d'ouvrage de la surveillance de son système d'information réside principalement dans la gestion des accès. « Qui peut voir quoi ? ». C'est particulièrement sur ce point que ce mémoire présente une réelle plus-value dans un projet de mise en place d'une surveillance du système d'information globale et ouverte aux utilisateurs.

Outre la complexité technique, les enjeux que représente le fait d'ouvrir ce genre d'information aux utilisateurs sont importants. En effet, donner des informations de disponibilité d'un système à un utilisateur, c'est lui donner un outil factuel qu'il peut utiliser pour critiquer la qualité des services rendus par le centre informatique.

Ce mémoire a pour objectif de mener une réflexion sur la possibilité d'ouvrir les données

de disponibilité aux utilisateurs, en définissant dans un premier temps un modèle conceptuel de surveillance du SI. Ce modèle s'appuie sur le référentiel des applications (CMDB[1]) ainsi que sur le référentiel de gestion des accès, dans un contexte tel que celui de l'État de Genève. Ce contexte se retrouve dans la plupart des grandes entreprises. Le but de ce modèle est de s'intégrer dans les processus ITIL[2] que le CTI met en place depuis 2007.

Dans un deuxième temps, une preuve du concept de ce modèle a été mis en application dans l'environnement du CTI, afin de démontrer la faisabilité d'un projet d'ouverture de la surveillance aux utilisateurs à large échelle.

Pour réaliser ce mémoire, j'ai collaboré avec les acteurs œuvrant à la mise en place d'une solution de surveillance globale à l'État de Genève (cf. sous-section *4.1.2 - Le projet ARGOS*). Le projet de surveillance d'applications est un projet de longue date qui a mis du temps à trouver sa direction actuelle. Ce projet a débouché sur un appel d'offres pour une solution qui devrait répondre à un plus large besoin. Contrairement à la vision de ce mémoire, il n'est pas question pour ce projet d'être ouvert à la maîtrise d'ouvrage.

Depuis que j'ai soumis la proposition du sujet de ce mémoire en automne 2009, un deuxième projet complémentaire (cf. sous-section *4.1.3 - Le projet SI-Alert*) de surveillance des applications a été lancé au sein du CTI. Ce second projet a été mis en place pour que le service Pilotage ait une visibilité globale du statut des applications critiques, afin de pouvoir contacter les équipes concernées en cas d'incident. Quelques tableaux de bord ont été ouverts à certains chefs de projets du développement. Ce projet est arrivé à terme. Le prototype réalisé dans le cadre de ce mémoire propose un complément à ce qui a été mis en place dans le cadre du projet SI-Alert.

[1] CMDB : (Configuration Management DataBase) est un référentiel (défini dans ITIL) des composants d'un système d'information. La CMDB définit le détail de chaque composant et les relations entre eux. Un composant peut être vu comme un composant informatique, un processus ou un utilisateur du système d'information.

[2] ITIL : Information Technology Infrastructure Library, est un ensemble d'ouvrage recensant les bonnes pratiques pour la gestion de services informatiques. La section *2.5 - La* surveillance applicative en relation avec ITIL est consacré à définir les bases d'ITIL

À ce jour, le CTI et la MOA ne possèdent pas de véritables outils de surveillance globale de l'état du système d'information. Le travail important qui a été fait jusque-là permet de surveiller d'un côté les composants techniques et d'un autre de faire des tests applicatifs.

Ce mémoire est composé de quatre chapitres : une définition du sujet, une analyse détaillée de la problématique de la surveillance d'un système d'information, un modèle conceptuel d'une méta-surveillance d'un système d'information qui s'appuie sur les référentiels existants de l'entreprise, et pour terminer une preuve du concept pour démontrer l'application du modèle à l'infrastructure du CTI, en tenant compte des contraintes existantes.

1 Définition

Que signifie « Doter la maitrise d'ouvrage d'une surveillance stratégique en temps réel de la disponibilité de son système d'information » ? La **surveillance** est une opération qui consiste à suivre en temps réel le fonctionnement d'un système ou d'un processus afin d'en détecter une anomalie. On peut ainsi en retracer la disponibilité du composant surveillé. L'état de **disponibilité** d'une ressource est la capacité de l'utiliser lorsqu'on en a besoin. ITIL définit la notion de disponibilité de la façon suivante : « *capacité d'un service informatique à produire sa fonction à un moment donné et sur une certaine période de temps (se présente sous la forme d'un pourcentage)* ». ITIL ne définit donc pas uniquement la disponibilité comme un état à un instant *t*, mais comme une mesure sur une période. Cela permet notamment de calculer le taux de couverture d'un « Service Level Agreement » (SLA)[3].

Une surveillance **stratégique** consiste en un processus de surveillance permettant de tirer parti des informations résultantes sur le long terme. Contrairement à une information temporaire qui donne l'état d'un système à un instant donné. Il est possible de stocker les informations capturées en temps réel, et d'utiliser ces informations à des fins d'anticipation de pannes, de prévision d'évolution du système, de déclenchement automatique de processus, ou encore de mesurer si l'infrastructure correspond bien aux besoins.

La notion de **temps réel** utilisée dans ce sens peut sembler abusive, car il ne s'agit pas d'un procédé de surveillance visant à déclencher une alerte de manière instantanée, mais d'un procédé de surveillance adapté à l'évolution du système contrôlé.

Le temps réel dans notre cas consiste à suivre de manière régulière l'ensemble de la disponibilité du système d'information surveillé. C'est-à-dire que les informations d'indisponibilité et de retour à la normale sont remontées à l'utilisateur de manière automatique et avec un délai aussi court que possible.

[3] SLA – Service Level Agreement, est un accord entre un fournisseur de services du centre informatique et un client permettant de définir le niveau de service attendu.

Surveiller un **système d'information** (SI) consiste à surveiller la disponibilité du système informatique, des applications, ainsi que contrôler la cohérence des données et des flux d'information échangés entre les applications. Cette approche permet d'avoir une vision globale de l'ensemble des éléments dont est constitué un système d'information.

La **maîtrise d'ouvrage** représente le client du système d'information. Ce terme sous-entend l'ensemble des personnes qui sont utilisatrices de ce système. Habituellement, les données de disponibilité d'un système ne sont pas accessibles par les utilisateurs. C'est en partant de cette constatation que l'on peut mieux se rendre compte de l'importance du sujet traité ici.

Le fait de rendre publique ce type d'information impose une parfaite maîtrise de la gestion des accès et de la définition des tableaux de bord publiés. Il est nécessaire de définir clairement quelles informations sont utiles à l'utilisateur pour avoir une vision de la disponibilité des outils informatiques qu'il utilise. Plus précisément, la seule information qui les concerne directement est la disponibilité instantanée des applications auxquelles il a accès.

La continuité des activités d'une entreprise dépend de la disponibilité des informations et de la possibilité de les traiter. L'interruption de l'accès à une ressource ou une information peut rapidement devenir une menace pour la pérennité de l'entreprise. La détection d'une telle anomalie permet la réduction des risques d'interruption des activités de l'entreprise en cas d'incident. Il existe une classification des ressources, afin d'évaluer l'impact en cas d'indisponibilité. La classification de la disponibilité s'évalue généralement d'après la gravité des pertes de profits ou de productivité qui résulteront de l'indisponibilité. Cela peut aussi être en fonction d'un risque physique sur les personnes, comme les services d'urgence. Pour déterminer à quelle catégorie il faut rattacher un risque, la société Harvard Research Group a publié un tableau avec une échelle de 6 niveaux (cf. *Annexe A*). Le premier niveau (AEC-0) considère que l'application peut être arrêtée en tout temps, sans problème de perte d'intégrité des données ou de dysfonctionnement du SI. Le dernier niveau (AEC-5) quant à lui correspond à une application qui doit continuer à fonctionner quelles que soient les circonstances.

On constate que l'activité économique des entreprises tend vers un fonctionnement 24h/24 et 7j/7. Ceci est notamment dû à la mondialisation et au commerce électronique qui impliquent des besoins élevés de flexibilité et de rentabilité. Il est courant aujourd'hui qu'un service pour un client soit fourni par une association de plusieurs partenaires, comme la réservation d'un voyage qui implique un vol en avion, la réservation d'un hôtel et la réservation d'une voiture. Si un des composants ne fonctionne plus, cela bloque toute la chaîne. Cela impact donc l'ensemble des fournisseurs associés à produire un service.

La surveillance de la disponibilité des composants et des applications se fait dans toute moyenne ou grande entreprise. C'est la seule méthode pour déceler un incident qui pourrait ne pas encore avoir impacté les utilisateurs. Mais il s'agit principalement d'utiliser ces données pour une meilleure compréhension de l'incident (couverture géographique, composant en erreur) et comme aide au diagnostic en faisant de la corrélation d'événement quand un utilisateur déclare un incident au Centre de Service[4]. C'est aussi une manière de faire ressortir des informations comme le taux d'occupation des serveurs, la congestion du réseau ou le contrôle de disponibilité des applications distantes. Ainsi, il est possible de prévenir le risque d'une interruption trop longue qui pourrait être préjudiciable pour l'entreprise. Néanmoins, si la nécessité d'une telle démarche n'est plus à démontrer, il est encore rare que l'ensemble du SI d'une entreprise soit surveillé, soit par manque de moyens, soit par difficulté à mettre en place une telle démarche.

Le contexte à l'État de Genève est particulier. Il y existe de nombreuses d'applications (environ 2000), qui tournent sur environ 2500 serveurs (virtuels et physiques). Toutes les applications ne sont pas référencées ni gérées par les mêmes personnes. Les applications ont des architectures techniques très hétérogènes, des types de serveurs, des langages de développement et des protocoles de communications différents. En fonction des applications, les techniques de développement, les modes de sécurisation ou les bases de

[4] Centre de Service : département de l'entreprise qui assure le service d'assistance informatique du processus Exploitation des Services selon ITIL.

données peuvent aussi être divers. De plus, la frontière entre les responsabilités techniques et métiers est loin d'être définie de la même façon partout.

Certaines applications sont sensibles alors que d'autres non. De plus, il y a un contexte politique à ajouter à la surveillance d'application, puisqu'il y a sept départements différents, qui ont chacun des règlements et des responsabilités spécifiques ainsi que des niveaux de criticité différents. À ce jour, presque chaque brique de l'architecture du SI est déjà surveillée par les équipes exploitantes, mais la mise en cohérence des résultats des surveillances n'est pas faite. Ainsi, à ce jour si un composant tombe en panne, il n'existe aucun moyen automatique d'annoncer à l'ensemble des services informatiques la liste des applications impactées. Il n'est alors pas envisageable de prévenir les utilisateurs concernés.

Une CMDB permet de connaitre les dépendances entre les applications, ainsi que les utilisateurs impactés par un composant défaillant. La mise en place d'un système de surveillance corrélé avec une CMDB permet de lever une alerte lorsqu'un composant dysfonctionne et de compléter les informations avec les composants de la chaine qui seraient impactés. Cette mise en place serait rassurante pour l'ensemble de l'organisation, d'une part, d'un point de vue informatique, car cela permettrait d'être fortement proactif sur les incidents, et d'autre part pour chaque utilisateur afin de s'assurer de la disponibilité des services qui lui sont offerts. La mise en activité d'une CMDB est prévue pour début 2012 au CTI.

2 La surveillance des systèmes d'information

La notion de surveillance d'un système d'information, quel qu'il soit, intervient lorsque l'indisponibilité de ce système pose un problème, et qu'il est nécessaire d'avoir une réaction aussi efficace que possible pour un retour à la normale dans les meilleurs délais, selon ce qui est défini dans les SLA. Cela peut être pour un problème de coupure de fonctionnement qui engendrerait une perte de données, de temps ou d'argent. Dans la nomenclature ITIL, il s'agit de « Availability Management ». Cela peut aussi simplement permettre d'anticiper l'achat de matériel supplémentaire. La notion d'anticipation de la scalabilité du système informatique fait partie du processus « Capacity planning » dans ITIL. La mise en place d'une méta-surveillance peut permettre de détecter des problèmes de congestion du réseau ou de ressource limitée et apporte ainsi une solution au besoin d'anticipation.

2.1 Faut-il surveiller l'informatique ou l'information ?

Tout au long de ce mémoire, il est plusieurs fois abordé le problème de la catégorisation de tel ou tel élément du système informatique. En effet, dans un processus de mise en production d'une application et de la mise en place de la surveillance de cette application, un élément peut être de type composant d'infrastructure ou de type service applicatif. C'est le cas d'un serveur web Apache[5] par exemple. Si celui-ci peut être considéré comme une application qui tourne sur un serveur physique, on parle couramment de « serveur Apache ». Du point de vue de l'ingénieur système, cette situation est considérée comme une application qui s'installe sur un serveur d'infrastructure. Du point de vue du chef de projet dont la mission est de mettre en place une nouvelle application métier, le serveur Apache fait partie de l'infrastructure, au même titre qu'un serveur physique. En effet, un serveur physique seul ne lui sert à rien, c'est pourquoi l'infrastructure est vue alors comme un tout.

[5] Serveur Apache : serveur HTTP libre, le plus utilisé sur le web

Afin de définir ce point pour la suite de ce mémoire, tout élément qui concerne l'utilisateur final est considéré comme une « application » ou un « service applicatif ». Et tout ce qui concerne les ingénieurs, ou plus globalement ce que le centre informatique met à disposition comme infrastructure d'un point de vue technique, est considéré comme un « composant ». De fait, les notions d' « application » et de « composant » seront traitées différemment au cours de ce mémoire. Tous les éléments, applicatifs ou composants, peuvent être surveillés.

Il existe deux types de surveillance, celle du :

- système informatique
- système d'information

Le premier type est lié à tout ce qui touche l'infrastructure, donc les composants, alors que le second concerne les services métiers. Cela inclut le contrôle de la qualité des données et des flux d'information entre les applications. Quand on parle de surveillance du système d'information, cela englobe inévitablement le système informatique, puisque le système d'information en est dépendant.

Selon certaines définitions, le « système d'information » inclut également les processus non informatisés, ainsi que les collaborateurs de l'entreprise. La notion de « système d'information » utilisée dans ce mémoire se focalise à l'information numérique.

2.1.1 Surveillance de l'infrastructure

La première partie à surveiller concerne tout ce qui est espace disque, température du processeur, occupation de la mémoire, disponibilité des machines, fonctionnement des serveurs de bases de données, les annuaires, les composants transversaux et tout ce qui se rapporte à l'infrastructure. Cela consiste à détecter un problème de type matériel ou service[6], indépendamment des applications métiers qui peuvent tourner sur les serveurs. C'est une surveillance technique.

[6] Service : dans ce contexte, il s'agit d'une fonctionnalité fournie par un composant pour assurer une tâche particulière.

Les tests permettant de faire cette surveillance technique sont communs à l'ensemble des composants du même type. Ainsi, il n'est pas nécessaire de redévelopper les tests techniques à l'ajout d'un nouveau serveur.

2.1.2 Surveillance des applications

La deuxième partie consiste à surveiller la disponibilité des applications, c'est ainsi qu'on vérifie qu'un service applicatif fonctionne correctement, et que les flux métiers fonctionnent entre les systèmes et ceci indépendamment de l'état de l'infrastructure. Ce que l'on nomme service applicatif peut être composé d'une multitude d'applications ou services qui tournent sur un serveur et qui constituent le système d'information. C'est une surveillance métier.

Pour surveiller des applications, cela requiert de définir des cas de tests métiers, qui sont spécifiques à chaque écran de l'application. Un écran accessible avec différents niveaux de responsabilités devra être testé avec autant de scénarios qu'il y a de possibilités.

2.2 Évolution du besoin de surveillance

2.2.1 La surveillance est-elle un besoin nouveau ?

La surveillance a considérablement évolué ces vingt dernières années. Avant, le système informatique des entreprises reposait sur un mainframe, application monolithique dont l'architecture était composée d'éléments propriétaires. Les postes utilisateurs n'avaient aucune complexité et seule l'unité centrale, contenant les données et les traitements métiers, ainsi que le réseau devaient être surveillés. La surveillance consistait à contrôler les lignes de log qui défilaient à l'écran. Lorsqu'une anomalie survenait, elle était détectée rapidement ou annoncée par les utilisateurs. Car bien souvent une anomalie sur un mainframe impactait un nombre important d'utilisateurs.

L'arrivée du protocole IP a augmenté la complexité des architectures réseaux en entreprise, c'est pourquoi les premiers composants à avoir été surveillés avec un système

dédié à la surveillance ont été les éléments réseau. Puis, avec l'arrivée d'Internet, chaque fournisseur d'accès à Internet se devait de surveiller la qualité des services fournis.

A ce jour, l'informatique est un monde distribué, où chaque composant offre une fonctionnalité spécifique. Les composants communiquent entre eux, et si l'un d'eux n'est plus accessible, c'est l'ensemble de l'architecture qui s'en trouve affaiblie. C'est pourquoi le besoin de surveillance est devenu capitale.

2.2.2 La surveillance s'applique-t-elle à tout le monde ?

2.2.2.1 *Le cas d'une PME*

Dans une PME, on retrouve en règle générale un collaborateur spécialisé en informatique, mais ce n'est pas une tâche qu'il occupe à plein temps. Il est courant qu'une PME fasse appel à une société informatique externe pour gérer son parc informatique. Une PME n'a guère besoin de solutions industrielles de surveillance de son parc informatique. Pour une entreprise de cette taille, les seules entités à être centralisées sont les données, le serveur mails et bien sûr les sauvegardes. Un outil de sauvegarde un tant soit peu sérieux, contrôle de manière automatique le bon déroulement de la sauvegarde, et alerte automatiquement un administrateur système en cas de problème. C'est suffisant.

2.2.2.2 *Une entreprise plus importante possède son propre service informatique*

Quand on passe à une taille d'entreprise supérieure, la gestion d'un parc informatique devient réellement plus complexe. En règle générale, un service informatique est créé au sein de l'organisation. Ce service informatique est chargé d'installer, de gérer et de veiller à ce que l'ensemble du système d'information fonctionne. Il est évident que si une partie du système ne fonctionne plus, cela peut concerner plusieurs dizaines de personnes et ainsi avoir un impact important sur l'entreprise. Cet impact peut être financier, médiatique, voire politique. Cela peut même être un risque pour la pérennité de l'entreprise. Dans tous les cas, c'est une préoccupation pour l'entreprise qui doit gérer les risques

d'indisponibilité ou de performance du système d'information et qui a une responsabilité vis-à-vis de ses clients.

Pour une société de taille moyenne, ce qui importe principalement c'est que le service informatique puisse intervenir dans des délais aussi courts que possible, voire qu'il puisse, dans la mesure du possible, anticiper un problème. En règle générale, il n'y a pas encore de service d'assistance dédié aux utilisateurs, et le service informatique assume lui-même cette tâche.

2.2.2.3 *Quand une grande organisation dépend de son système d'information*

Dans une grande organisation, le service informatique joue un rôle essentiel pour permettre à l'entreprise d'offrir à ses clients un service de qualité. Il devient alors crucial de surveiller le système d'information et de fonctionner avec une architecture redondante, pour qu'en cas de panne d'un composant, les utilisateurs ne soient, si possible, pas impactés. Il est alors obligatoire d'avoir un système de surveillance performant, afin de détecter une anomalie avant même que l'utilisateur ne s'en aperçoive, puisque des composants de secours continuent à fonctionner. Un système de surveillance est complexe à mettre en place. Le niveau d'alerte à définir doit correspondre aux besoins de l'entreprise. Si on considère qu'une application n'est pas critique, il n'est pas nécessaire de lever des alertes urgentes au détriment d'une autre application. La gestion des priorités devient importante.

2.2.3 Adopter un système de surveillance centralisé

La problématique d'un centre informatique aussi important que celui du CTI est que chaque service a mis en place une surveillance de ses composants, mais que malheureusement aucune mise en commun n'est réalisée. Il n'est donc pas possible de définir un accord entre les fournisseurs de service (appelé OLA[7] dans la nomenclature

[7] Operational Level Agreement : Accord entre les fournisseurs de services d'un même centre informatique, qui définit le niveau d'exigences entre les services et composants informatiques.

ITIL), ni de corréler les alertes, et encore moins de définir une disponibilité globale pour l'utilisateur. On ne connait pas l'état du système en un coup d'œil.

L'avantage d'avoir un système de surveillance centralisé est la maitrise complète de ce qui est surveillé. La définition des alertes se gère alors de manière cohérente, selon des règles communes à l'ensemble des éléments surveillés. Un système mal paramétré et qui lève trop d'alertes, va noyer les administrateurs système dans un flot d'informations important et va limiter leur capacité à réagir. Ils seront obligés de trier les alertes et de prioriser celles-ci avant d'intervenir, voire n'y prêteront plus attention : ce n'est pas acceptable. Une bonne analyse, avec une définition du niveau de service souhaité par l'entreprise avec l'établissement d'OLA entre les services internes et de SLA avec le client est nécessaire. Cette mise en place requiert des moyens financiers. La surveillance a un coût qui n'est pas négligeable, et c'est malheureusement souvent là que le bât blesse. Il est toujours difficile de faire le choix d'investir de l'argent dans une situation qu'on ne souhaite pas voir se produire. Il faut considérer cela comme une assurance pour l'entreprise. Il s'agit de couvrir un risque que l'on ne souhaite pas voir apparaître dans son entreprise.

Il arrive souvent qu'un utilisateur considère son application comme critique, car il ne peut plus travailler sans elle. Ce que cet utilisateur ne réalise pas, c'est que d'autres applications permettent à des centaines, voire des milliers de personnes de travailler. Certaines de ces applications sont également utilisées en interaction directe avec les clients qui se présentent aux guichets. Si la surveillance du système d'information n'est pas faite de manière centralisée, il est possible qu'une petite application profite de la mise en place d'une surveillance, alors que d'autres applications critiques n'en seraient pas encore pourvues. Avec un système centralisé, cette méprise ne pourrait pas se produire.

Il est évidemment nécessaire de bien connaître son système d'information et de définir une priorité d'intégration avant de commencer la mise en place d'un projet de surveillance centralisé. La tâche de référencement de l'ensemble des applications utilisées quotidiennement par les utilisateurs n'est pas une action aussi simple à mener.

Avoir la cartographie complète des applications constituantes du SI est déjà une tâche compliquée. Il faut ajouter à cela la catégorisation de chaque application (voir Annexe A) et la définition des critères de priorités (voir la sous-section *3.4.4 - Réflexion sur la pondération d'indicateur*).

La réussite d'un projet de mise en place d'une surveillance centralisée est conditionnée par la maturité de l'organisation.

2.3 Problématique des alertes

Comment lever une alerte qui soit cohérente ? Voici un exemple de cas réel rencontré dans une grande entreprise : lorsqu'une chaine de sauvegardes[8] sur les serveurs hébergeant les bases de données s'interrompt, cela génère une alerte et un ticket d'incident[9] pour chacune des applications métiers faisant référence à ces bases de données, bien que les applications ne soient en réalité pas impactées. Les tickets sont envoyés de manière automatique à l'équipe d'intégration. Comme chaque ticket concerne des applications métiers différentes, il y a de grandes chances que ce soit une personne différente qui reçoive le ticket de panne à traiter. Chacune de ces personnes va prendre le temps d'analyser, contrôler et tester son application pour comprendre le problème. Chacun finit par déduire que c'est la chaine de sauvegardes qui a provoqué cette erreur, et complète l'information du ticket d'incident avec son propre diagnostic et le transfère à l'équipe qui s'occupe de la chaine de sauvegardes. Cette équipe, au bout d'un certain temps, constate que le problème touche plusieurs applications et reçoit en retour chaque ticket de panne envoyé par les différents intégrateurs. Il est possible que la panne de la chaîne de sauvegardes a elle aussi généré une erreur qui a directement été envoyée à la bonne équipe. Cette panne est peut-être déjà corrigée, alors même que les intégrateurs n'ont pas encore fini d'analyser les premiers incidents reçus.

[8] Ensemble de sauvegardes qui sont faites de manière automatique sur plusieurs serveurs.

[9] Un ticket d'incident, ou ticket de panne est une fiche électronique qui décrit le problème énoncé et qui permet d'avoir un suivi du traitement de la panne jusqu'à sa résolution. En règle générale, le système qui permet de gérer les tickets est intégré à un workflow qui permet de documenter et de faire parvenir automatiquement le ticket d'incident au bon service. Lorsqu'un ticket est fermé, l'utilisateur ayant ouvert le ticket d'incident est automatiquement informé de sa résolution.

Avec une méta-surveillance qui intègre et fait une synthèse des différents systèmes de surveillance, cette perte de temps et d'argent peut être évitée. Si cela avait été le cas dans notre exemple, l'intégrateur aurait vu immédiatement le statut en erreur de la chaine de sauvegardes. Il pourrait même consulter l'ensemble des applications qui dépendent de cette chaine de sauvegardes, et ainsi être proactif et prévenir ses collègues.

Un autre exemple rencontré dans une entreprise importante : un composant utilisé par deux applications jugées critiques et stratégiques pour l'entreprise est partiellement tombé en panne. C'est-à-dire que certaines fonctionnalités étaient inopérantes. Un intégrateur responsable d'une des applications en lien avec ce composant a vu la panne et s'en est occupé. Voulant bien faire, il a contacté le centre d'assistance afin d'ouvrir un ticket pour avoir une trace de cet incident critique. Comme il s'en occupait, le ticket d'incident lui a été directement envoyé. Ce que cette personne ignorait, c'est que ce composant était également utilisé par une deuxième application. Voulant rétablir le lien entre le composant et son application, il a demandé à ce que l'on redémarre le serveur hébergeant ce composant. Cela a généré des perturbations importantes pour la deuxième application, elle aussi stratégique. Le fait que cette application devienne inactive suite à l'action d'un ingénieur du centre informatique a créé une polémique bien plus importante que le fait d'avoir un composant dont certaines fonctionnalités étaient indisponibles.

S'il y avait eu un dispositif d'alerte centralisé, lié avec une notion de dépendance entre les applications et les composants, cela aurait été découvert immédiatement, et cette erreur d'un ingénieur qui pensait bien faire n'aurait jamais eu lieu. C'est pourquoi il est indispensable que le système de surveillance soit couplé à une CMDB. Certains produits permettent d'avoir une découverte automatique de l'infrastructure soutenant une application. Si l'infrastructure est importante, ce genre de solution peut faire gagner un temps certain.

Un système d'alerte est particulièrement utile lorsqu'une application tombe en panne durant la nuit. L'ingénieur recevant l'alerte a alors théoriquement le temps d'intervenir

avant la reprise du travail le matin suivant. Car il n'y a rien de plus désagréable pour un utilisateur que d'avoir son application qui ne soit pas accessible en arrivant de bonne heure le matin au travail, et de devoir attendre que les ingénieurs identifient et corrigent la panne pour commencer à travailler.

Le dernier cas soulevé est celui d'un composant en cluster. Si l'un des clusters tombe en panne, alors on doit pouvoir le rétablir, sans même que l'utilisateur ne le constate. Cela implique de séparer les tests applicatifs des tests infrastructure, et de lever des alertes pour chaque composant qui pourrait tomber en panne, indépendamment du service applicatif. Un test applicatif seul, il n'aurait jamais détecté qu'un composant en cluster puisse être en panne, puisqu'un autre composant aurait pris la relève.

2.4 Objectif : Haute disponibilité

Il est nécessaire de prévoir des solutions en cas d'arrêt non prévu de l'un ou plusieurs composants ou systèmes. Le risque à assumer est différent en fonction de l'activité de l'entreprise, mais les règles de base s'appliquent à chacun. Pour certaines entreprises, comme celles du secteur bancaire, le besoin de disponibilité est tel que pour une partie de leurs activités, il est prévu d'avoir des salles complètement redondantes. Ces salles sont appelées salles blanches, elles se situent en général sur un autre site, à plusieurs dizaines de kilomètres. Car en cas de coupure de courant généralisé, il est indispensable que cette deuxième salle se trouve sur un réseau électrique en tout point différent du premier.

Lorsqu'on parle de haute disponibilité, il est en fait question de garantie de fonctionnement d'un système. Cette garantie se calcule en heures par jour et en jours par année. À chaque mise à jour d'un composant, que ce soit matériel ou applicatif, s'il est nécessaire de redémarrer le serveur, ou même un service nécessaire à faire fonctionner l'application, alors ce redémarrage diminue le temps total de disponibilité. Les interruptions de service planifiées dans les fenêtres de maintenance annoncées à l'avance ne sont pas à prendre en compte dans le calcul de la disponibilité.

La disponibilité se définit à l'aide d'un pourcentage de temps où le système est actif par rapport au temps où le système est éteint. L'objectif, presque impossible à réaliser tellement cela serait coûteux, est d'avoir un taux de disponibilité de 100 %. Un système ayant un taux de disponibilité de 99 %, ce qui pourrait paraître satisfaisant, donne en réalité, sur une base de 24 heures sur 24 et 365 jours par an, une indisponibilité de 3,65 jours par an. Si cela se produit la nuit, alors que le système n'est pas utilisé, c'est probablement transparent pour l'utilisateur. Mais si des pannes se répètent régulièrement pour une heure ou deux durant la journée, les 3,65 jours d'indisponibilité par an vont devenir un réel problème pour les utilisateurs. C'est pourquoi dans les SLA, il est défini un horaire journalier durant lequel le système doit fonctionner. Par exemple entre 7h et 20h le système doit être disponible, s'il y a des coupures en dehors de ces heures, alors cela n'a pas d'importance et cela n'intervient plus dans le calcul de la disponibilité.

Tableau I. Correspondance de disponibilité en % sur une année

Disponibilité en %	Durée d'interruption maximale par an
96	14,6 jours (donc > 2 semaines)
98	7,3 jours (donc > 1 semaine)
99	3,65 jours
99.9	8,76 heures
99.99	52,56 minutes
99.999	5,26 minutes
99.9999	31,5 secondes
99.99999	3,15 secondes

Pour améliorer la disponibilité d'un système, certains éléments doivent être redondants. Pour réaliser cela, il est nécessaire d'investir dans la redondance de l'ensemble des composants, ce qui coûte cher. C'est donc un problème de coût qui va limiter la mise en place d'une architecture haute disponibilité.

Il faut aussi remarquer que pour fournir une application à l'utilisateur final, de nombreux composants ayant une dépendance entre eux sont nécessaires. Si l'on considère qu'il y a un serveur à 99 % de disponibilité, une infrastructure réseau à 99,9 % et un applicatif à

97 %, la disponibilité totale pour l'utilisateur n'est pas une pondération des trois niveaux, mais est dégressive. Cela signifie qu'il y a un risque que l'application finale soit à 95,9 % d'indisponibilité, soit près de 15 jours par an.

2.4.1 Calcul de la disponibilité

On définit la disponibilité d'un service en fonction de la disponibilité de ses composants. L'architecture des composants influence la disponibilité. Des éléments en série baissent la qualité de la disponibilité, alors que des éléments en parallèle augmentent la disponibilité.

Disponibilité en série = $D_1 \times D_2 \times D_3 \times D_n$

Disponibilité en parallèle = $1 - ((1 - D_1) \times (1 - D_2) \times (1 - D_3) \times (1 - D_n))$

Disponibilité en parallèle pour un même composant = $1 - (1 - D)^n$

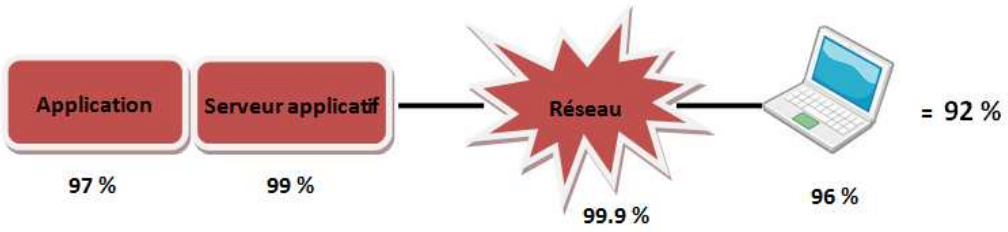

Figure n°1 - Éléments en série

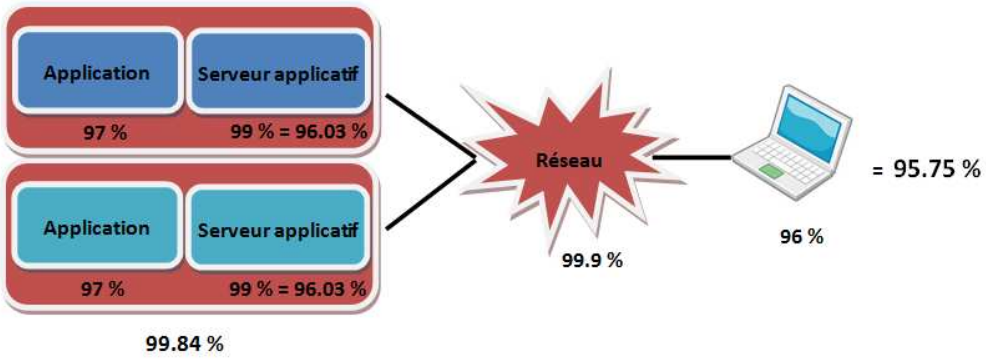

Figure n°2 - Éléments en parallèle

Afin de garantir un investissement financier raisonnable pour gérer la haute disponibilité, il est nécessaire de faire un choix des applications à mettre sur une infrastructure de haute disponibilité. Mettre tout le système informatique en haute disponibilité serait bien trop onéreux. En règle générale, les entreprises définissent 2 ou 3 niveaux de disponibilités différentes en fonction du risque. La société Harvard Research Group (HRG) a défini une classification sur six niveaux (cf. *Annexe A*).

Si l'on considère une classification sur trois niveaux, on aura ainsi un environnement de disponibilité standard pour les applications non critiques. Ces applications pouvant être interrompues sans causer de dommage à l'entreprise, elles ne seront donc pas redondantes. Il peut y avoir un environnement fiable, avec une partie redondante, mais sans que cela soit considéré comme de la haute disponibilité. Et finalement, l'environnement haute disponibilité implique que tous les serveurs (un par jeu de redondance) peuvent être coupés et redémarrés sans que cela impacte l'utilisateur. Les disques durs peuvent être en RAID 5[10], ce qui signifie qu'on peut simplement débrancher un disque, à chaud, et en rebrancher un vierge, toujours à chaud, sans que cela impacte l'utilisateur. Les données vont se resynchroniser de manière complètement automatique et autonome. Il existe également des systèmes d'adresse IP virtuelle, comme heartbeat[11], qui permettent de définir une adresse IP unique pour plusieurs serveurs. Si l'un d'eux tombe en panne, le système heartbeat détecte automatiquement l'adresse IP à stopper et renvoie les requêtes vers les serveurs qui répondent toujours.

Le fait d'élever le niveau de fiabilité génère d'autres problématiques :

- Les coûts matériels et logiciels augmentent, avec la redondance des équipements informatiques comme les serveurs et le réseau. Les charges d'exploitation et les dépenses énergétiques pour alimenter tous les composants redondants sont également plus élevées.

[10] RAID signifie « Redundant Array of Independent Disks ». Il s'agit de mettre plusieurs disques durs ensemble afin que l'information soit dupliquée. Si un disque tombe, l'information se trouve encore sur d'autres disques et peut ainsi être reconstituée. Il existe 7 niveaux de RAID (0 à 6) qui ont chacun leurs avantages et inconvénients. Le RAID 5 est le plus utilisé. La technologie RAID 5 permet notamment de débrancher un disque « à chaud » sans que cela interrompe le fonctionnement du serveur.

[11] Heartbeat est une architecture de haute disponibilité.

- La détection et l'identification de pannes deviennent plus complexes et nécessitent des compétences hautement qualifiées.
- Il est nécessaire de faire un test complet de toutes les procédures susceptibles d'affecter l'application. C'est délicat et complexe à tester en cas réel et peut générer des pannes imprévues.

Tester une infrastructure de haute disponibilité en cas réel est d'une grande complication. Car c'est justement les applications qui ne doivent jamais être arrêtées qui sont mises en haute disponibilité. Nous avons eu une expérience enrichissante au CTI lors d'une coupure de l'alimentation électrique de l'une des salles machines. Notre composant sécurité, qui est en haute disponibilité, a malgré tout été coupé dans de rares cas, car des serveurs qui font appel à notre composant n'ont pas tenu compte des changements effectués au niveau des DNS[12] (problème de cache DNS) et continuaient à appeler les serveurs hors service.

Résoudre un problème de ce type a demandé beaucoup d'effort, d'une part pour comprendre le problème et ensuite pour trouver une solution. La haute disponibilité coûte cher, et pour garantir une haute disponibilité proche des 100 %, chaque service doit investir du temps et de l'argent, à tous les niveaux. Effectuer un vrai test en coupant de manière réelle une salle machine est un risque que peu d'entreprises sont prêtes à prendre.

2.4.2 Cloud Computing, une révolution ?

Depuis quelques années, une technologie émergente, le Cloud Computing, répondra probablement à la problématique de haute disponibilité du système informatique. Le Cloud Computing est un concept de déportation de la puissance de calcul et de stockage sur des serveurs externes reliés entre eux et pouvant se trouver partout autour du

[12] Domain Name System : la résolution DNS est le fait de traduire une adresse de domaine en texte (ex. : www.google.ch) par son adresse IP physique (74.125.43.147). Un DNS peut être attaché à plusieurs adresses IP. Lorsqu'un serveur a fait la résolution DNS, il garde ensuite en cache l'adresse IP, ce qui peut poser problème si cette adresse disparaissait (comme lorsqu'un serveur est éteint).

monde. Ainsi les entreprises accèdent à leurs applications à travers Internet et n'ont plus besoin de gérer eux-mêmes des infrastructures importantes et complexes.

Le Cloud Computing est une solution aux problèmes de disponibilité, de sécurité et intègre une surveillance performante d'un tel système. Le principe du Cloud Computing est simple : offrir une prestation à la demande. Cela signifie que chaque entreprise est facturée selon ce qu'elle consomme. L'architecture du Cloud Computing devrait considérablement augmenter la qualité de la disponibilité, sans pour autant augmenter le coût d'une telle infrastructure.

Le Cloud Computing touche autant les sociétés multinationales qui cherchent une solution stable qui couvre une zone géographique importante, que les particuliers qui peuvent ainsi avoir un accès simplifié à une infrastructure pourtant complexe et de qualité à prix abordable.

2.5 La surveillance applicative en relation avec ITIL

ITIL, qui signifie « Information Technology Infrastructure Library », est une collection de livres qui recense, synthétise et détaille les meilleures pratiques pour une direction informatique fournisseur de services au sein de l'entreprise. L'objectif de cette approche est, une fois mise en place dans une organisation, que tous les informaticiens utilisent le même vocabulaire et les mêmes processus quel que soit le service informatique. Le but est de créer une cohérence et une synergie autour des processus du centre de production informatique.

ITIL aborde notamment les sujets suivants :

- Comment organiser un système d'information
- Comment améliorer l'efficacité du système d'information
- Comment réduire les risques
- Comment augmenter la qualité des services informatiques

Dans le cas de la surveillance du système d'information, il s'agit bien de réduire les

risques, ainsi qu'augmenter la qualité du service que peut attendre un client.

La création d'ITIL a été instaurée par le gouvernement britannique dans les années 1980. ITIL est devenu depuis un standard qui a su se développer et s'imposer dans toute l'Europe. ITIL pose les fondements de la gestion de services IT. L'informatique et ses technologies ne sont plus considérées comme un but en soi, mais comme un facteur de production critique pour toute l'entreprise. Ainsi, l'informatique doit pouvoir générer de la valeur et prouver sa contribution au succès de l'entreprise. L'approche orientée client est mise en avant dans ITIL, et c'est précisément le sujet de ce mémoire, offrir au client un service auquel il n'a que rarement accès. C'est-à-dire avoir une consultation de l'état de la disponibilité en temps réel de son système d'information. Il est à noter que peu de directions informatiques souhaitent mettre les données de disponibilité en temps réel à la disposition des utilisateurs.

ITIL en est à sa version 3 depuis l'été 2007. À chaque itération, des livres sont ajoutés à ceux qui existent déjà. Actuellement, les cinq livres de références sont : l'introduction au cycle de vie des services, la stratégie des services en général (Service Strategy), la conception des services (Service Design), la transition des services (Service Transition) et l'exploitation des services (Service Operation).

Figure n°3 - L'amélioration continue des services place le client au centre (Continual Service Improvement) (src : www.itilfrance.com)

Les tâches de surveillance sont spécifiées principalement dans « l'exploitation des services ». En cas d'alerte, cela va déclencher un processus de gestion des incidents qui peut influencer d'autres parties d'ITIL. L'exploitation de service est à considérer dans un rôle de gestion quotidienne et de gestion technique, elle est responsable de contrôler l'exécution et l'aboutissement des processus qui optimisent le coût et la qualité des services. Elle doit aussi permettre aux organisations métiers d'atteindre leurs objectifs, et elle est également responsable du bon fonctionnement des composants techniques qui constituent les services.

Pour atteindre ces objectifs, ITIL préconise d'impliquer dès la conception des services les équipes d'exploitation. Ainsi, cela permet de faire le lien entre les services à fournir et les infrastructures à assembler pour fournir le service. Cela amène également à lier la performance du service à celle des composants techniques associés.

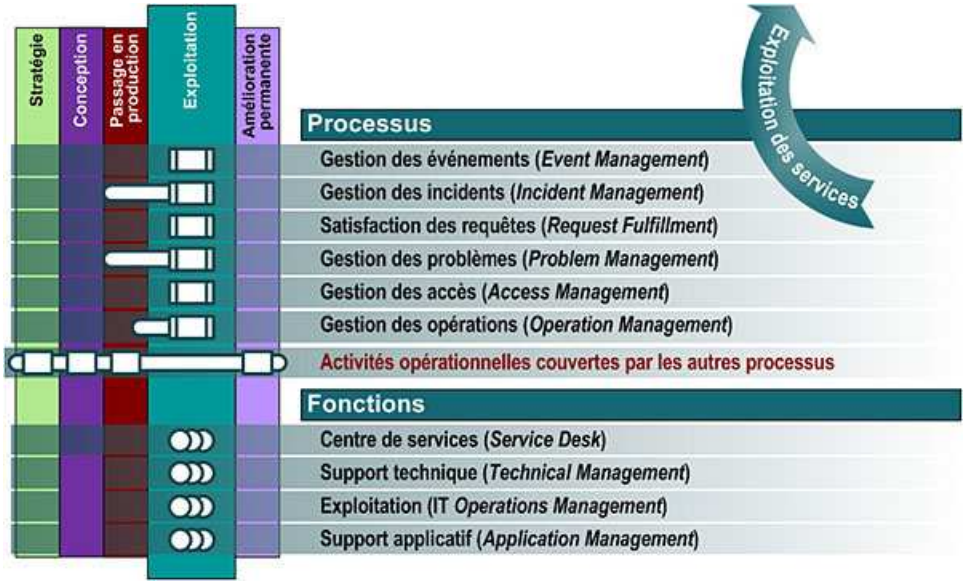

Figure n°4 - Les processus qui composent l'exploitation des services (src : www.itilfrance.com)

Le processus de gestion des événements est celui qui concerne le plus la surveillance du système d'information. Ce processus détecte les événements, leur donne une signification et détermine la réaction à adopter, qu'elle soit automatique ou humaine.

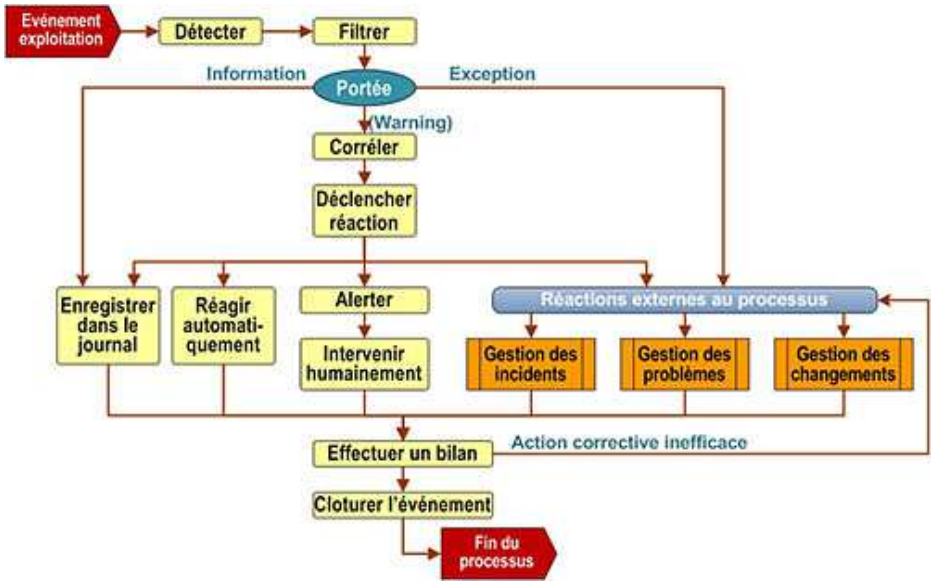

Figure n°5 - Processus de traitement d'un événement (src : www.itilfrance.com)

On constate qu'un événement qui est détecté va déclencher une série d'actions qui peut aboutir à la gestion des incidents, à la gestion des problèmes ou à la gestion des changements. La surveillance d'application n'a pas pour rôle d'aller s'immiscer dans la gestion des changements, mais il est clair qu'elle peut être une source d'information utile pour la prise de décision avant un changement de composant.

2.5.1 Gestion des incidents

Un incident est une interruption imprévue du fonctionnement ou une réduction de la qualité de service, celui-ci concerne les applications ou encore un des composants d'une application. En plus de détecter les événements et de trouver le meilleur moyen d'agir, la gestion des incidents doit aussi s'assurer du rétablissement de la fonction normale d'un service, comme défini dans les SLA.

2.5.2 Gestion des problèmes

Un problème est la cause inconnue de la source d'un incident ou d'une série d'incidents. L'objectif de la gestion des problèmes est de prévenir l'apparition des problèmes et des

incidents qui en découlent, d'éliminer les incidents récurrents et de minimiser l'impact des incidents imprévisibles. La surveillance et le contrôle sont des activités primordiales de l'exploitation des services et peuvent grandement aider à détecter la source de certains problèmes.

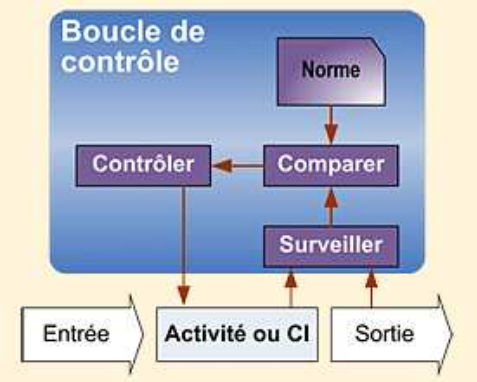

Figure n°6 - Boucle de contrôle d'un composant (src : www.itilfrance.com)

Surveiller et contrôler consiste à mettre en place des cycles permanents de surveillance, de génération de rapports et d'actions, ainsi que la mesure de la gestion des niveaux de services. Cela s'applique au niveau d'une application, d'un processus ou d'un composant de l'infrastructure. Le but est de détecter tout changement ou état dépassant une norme. Pour mesurer un changement, le seuil doit être défini et paramétré, ainsi que l'action qui va être déclenchée par le changement d'état. Cela s'applique autant à mesurer la disponibilité que la performance d'une application ou d'un composant.

ITIL parle de contrôle opérationnel lorsque ces contrôles sont effectués au niveau d'un composant. On s'inscrit alors dans une mesure d'assurance qualité, lorsque l'ensemble d'une chaine de composants, ou d'une application est contrôlé. La surveillance applicative, couplée à la surveillance de chaque composant est un outil puissant de l'assurance qualité.

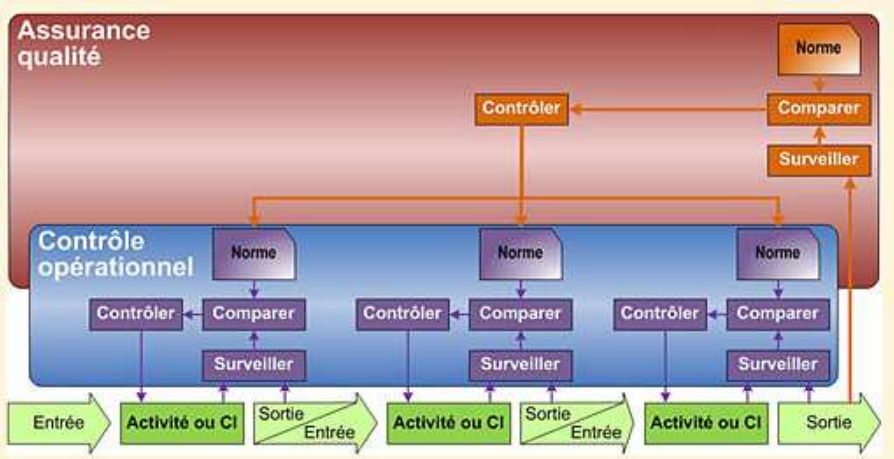

Figure n°7 - Processus d'assurance qualité intégrant les boucles de contrôle (src : www.itilfrance.com)

L'objectif intrinsèque de la mise en place d'une surveillance d'un système d'information est la gestion de la disponibilité. Mais la disponibilité n'est pas la seule chose à garantir pour satisfaire le client d'un système d'information. Six points clés sont à prendre en considération :

- **Disponibilité** : capacité d'un service informatique à produire sa fonction à un moment donné et sur une certaine période de temps (se présente sous la forme d'un pourcentage).
- **Fiabilité** : période durant laquelle le service n'a connu aucune défaillance (se présente sous la forme d'une durée).
- **Maintenabilité** : capacité d'un service informatique à rester ou à revenir sur son état opérationnel normal (la norme étant définie dans la convention de services).
- **Sécurité** : critères de confidentialité, d'intégrité et de disponibilité des informations associées à un service.
- **Aptitude au service** : description des contrats définis avec les fournisseurs pour assurer le niveau de disponibilité, de fiabilité et de maintenabilité.
- **Fonction métier vitale** : reflète les éléments critiques du processus métier supportés par un service informatique.

La disponibilité est le critère essentiel de ces six points et la surveillance apporte une mesure permettant de répondre à au moins une problématique de chacun de ces six critères. La disponibilité se mesure avec différentes métriques.

2.5.3 Métriques de disponibilité

Chacune de ces mesures intervient dans les accords entre chaque fournisseur de services, entre un fournisseur de service et son client ou encore avec les fournisseurs de services externes. Ces mesures sont prises uniquement grâce à la surveillance du SI et des composants. Ce sont des données cruciales pour déterminer le niveau et la qualité de service d'un centre informatique. Il existe trois accords différents en fonction des relations : niveaux de services, niveaux opérationnels, entités externes. Ils sont décrits en détail dans les paragraphes ci-dessous. Les métriques de disponibilités définies par ITIL sont les suivantes :

- **Fréquence des pannes** : par jour, par mois, par an.
- **Performance des restaurations** : temps de restauration et de redémarrage du Service après interruption.
- **MTBF** (Mean Time Between Failures) : temps moyen entre le redémarrage complet d'un Service et son interruption suivante.
- **MTBSI** (Mean Time Between System Incidents) : temps moyen entre deux pannes.
- **MTTR** (Mean Time To Repair) : temps moyen entre l'apparition d'un Incident et sa résolution.

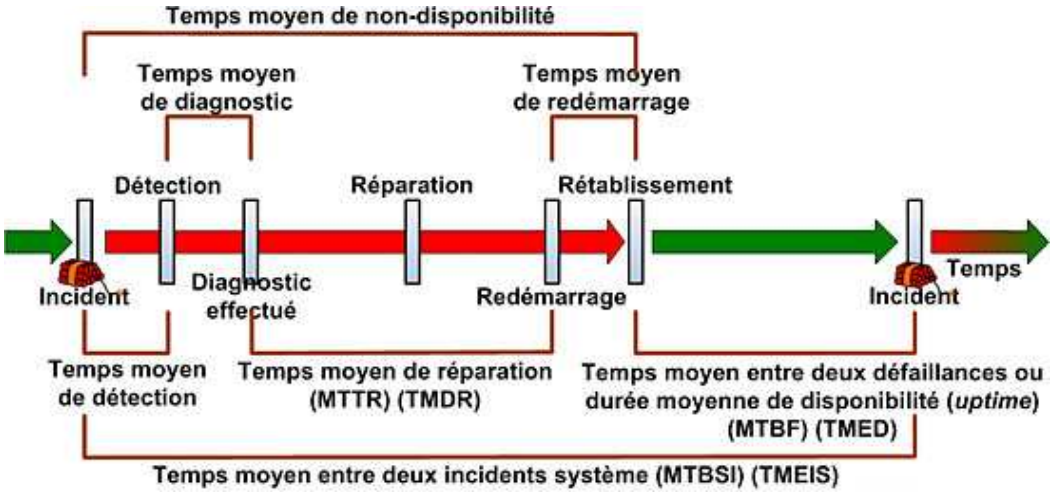

Figure n°8 - Schéma des métriques de disponibilité (src : www.itilfrance.com)

2.5.4 Accord sur les niveaux de services

Un accord entre un fournisseur de services du centre informatique et un client est appelé un SLA (Service Level Agreement). Le SLA décrit le service des technologies de l'information, documente les objectifs du niveau de services attendu et spécifie les responsabilités du fournisseur de service et du client. Un seul SLA peut couvrir plusieurs services ou plusieurs clients.

2.5.5 Accord sur les niveaux opérationnels

Un accord entre les fournisseurs de services d'un même centre informatique (entre le réseau et les serveurs par exemple) est appelé un OLA (Operational Level Agreement). Un OLA permet de définir le niveau d'exigences entres les services et composants qui vont au final, une fois mis bout à bout, fournir le service attendu par le client. Un SLA ne peut pas être plus exigeant que les OLAs passés entre les services informatiques dont il dépend.

2.5.6 Accord avec les entités externes

Un accord entre un fournisseur de services du centre informatique et une entité externe à l'organisation est appelé un UC (Underpinning Contract). Cela fonctionne selon les mêmes principes qu'un OLA, à l'exception du cadre juridique qui est différent, car il se signe avec des sociétés tierces.

3 La Méta-surveillance

3.1 Les enjeux d'une méta-surveillance

L'objectif n'est pas de réinventer ce qui existe déjà, ni réimplémenter un système de test complet. Il existe des logiciels qui font ça très bien. Certains, utilisés au CTI, sont présentés dans la section *3.7 - Les scénarios possibles pour une méta-surveillance*. Le but est de développer un modèle générique qui permette d'avoir une vue d'ensemble de la disponibilité des SI, quel que soit le niveau de granularité et quels que soient les types de systèmes informatiques concernés. Ce modèle doit apporter une solution pour améliorer le dialogue entre les équipes du centre informatique, afin d'améliorer la gestion des incidents, et surtout d'ouvrir certains tableaux de bord aux utilisateurs.

Ce modèle se compose de trois sous-systèmes :

- La définition des profils.
- La définition des relations et dépendances entre les applications et les composants.
- La définition des tests.

Chacun de ces sous-systèmes s'appuie sur des référentiels existants de l'entreprise :

- Le référentiel de la sécurité pour toute la gestion des accès.
- La CMDB est le référentiel des applications et des composants dont elles dépendent, ainsi que les liens entre chaque composant. Une CMDB recense également les composants qui sont en cluster[13].
- Le référentiel des tests qui regroupe tous les tests qui sont déjà effectués dans l'entreprise. Il se compose souvent d'un ensemble d'application de tests.

[13] Cluster : ensemble de serveurs, au minimum deux, garantissant une redondance entre eux pour assurer une continuité de service et/ou répartir la charge de calcul et/ou la charge réseau.

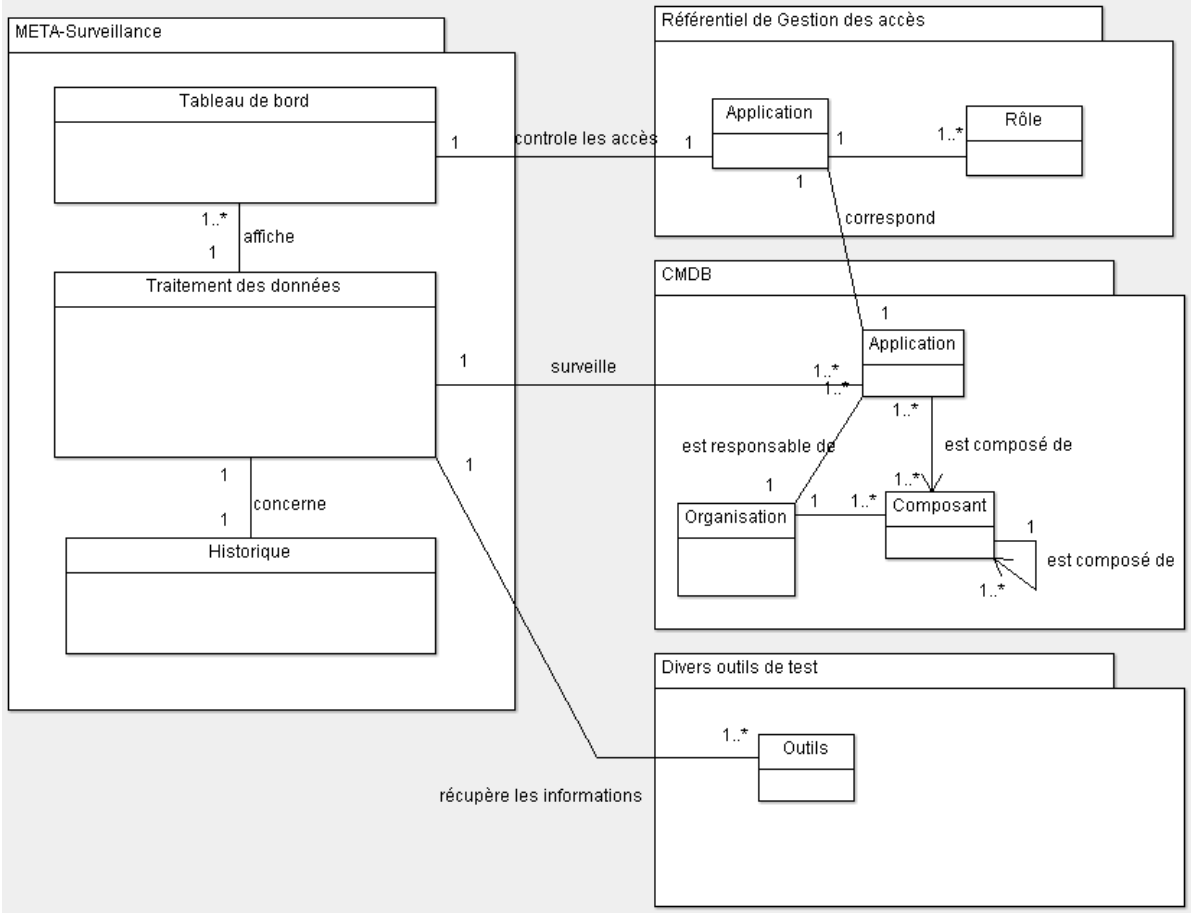

Figure n°9 - Diagramme de classe d'une méta-surveillance en relation avec les référentiels existants

Il est rare qu'une organisation ait déjà fait l'effort et l'investissement dans une solution unique et complète de test, et il existe probablement plusieurs solutions individuelles déjà en place. Le modèle développé doit pouvoir s'adapter à l'architecture de n'importe quelle entreprise en faisant le lien entre ces différents sous-systèmes. Une fois cette vue d'ensemble étudiée, j'aborderai dans un deuxième temps comment adapter ce modèle générique au contexte de l'État de Genève avec la réalisation d'un POC[14] fonctionnel.

La réussite d'un projet de surveillance globale au niveau d'une organisation comme le CTI permettrait de réduire les coûts et d'augmenter la satisfaction du client. La réduction des

[14] POC : Une preuve de concept (Proof Of Concept en anglais) est une réalisation courte ou incomplète d'une certaine méthode ou idée pour démontrer sa faisabilité.

coûts intervient principalement sur le temps gagné en analyse de panne et sur le temps passé par chaque service pour gérer individuellement une surveillance de son système.

L'ouverture cohérente et maîtrisée de l'information aux différents niveaux de responsabilité, ainsi qu'aux utilisateurs, permettrait d'ouvrir le dialogue entre chaque acteur concernant les contraintes et la complexité d'une infrastructure informatique, ce qui amène inévitablement à une discussion sur les coûts.

Finalement, une méta-surveillance est une garantie de qualité maîtrisée pour la direction du centre informatique.

3.2 Architecture de la surveillance

L'architecture doit répondre aux besoins d'une organisation ayant un parc applicatif très hétérogène, tant sur les technologies de développement, que les serveurs qui hébergent les applications et même dans l'architecture. Il existe des applications Web (java, PHP, Perl, .net), ou en client lourd (Java, Oracle, Access, FileMaker…), donc l'architecture d'un outil de surveillance doit comporter plusieurs « couches ». Il est commun d'avoir une structure en trois tiers :

1. L'interface pour l'utilisateur
2. La partie métier, traitement et corrélation des données
3. Le persistance ou stockage des données

Cela constitue la structure de base de l'architecture d'une solution de surveillance. À cela s'ajoutent les modules de déclenchement des alertes par SMS ou email, de génération de rapports ou encore les connecteurs à d'autres services.

Chacune de ces parties peut être installée sur un serveur dédié, notamment pour des raisons de performance ou de fiabilité.

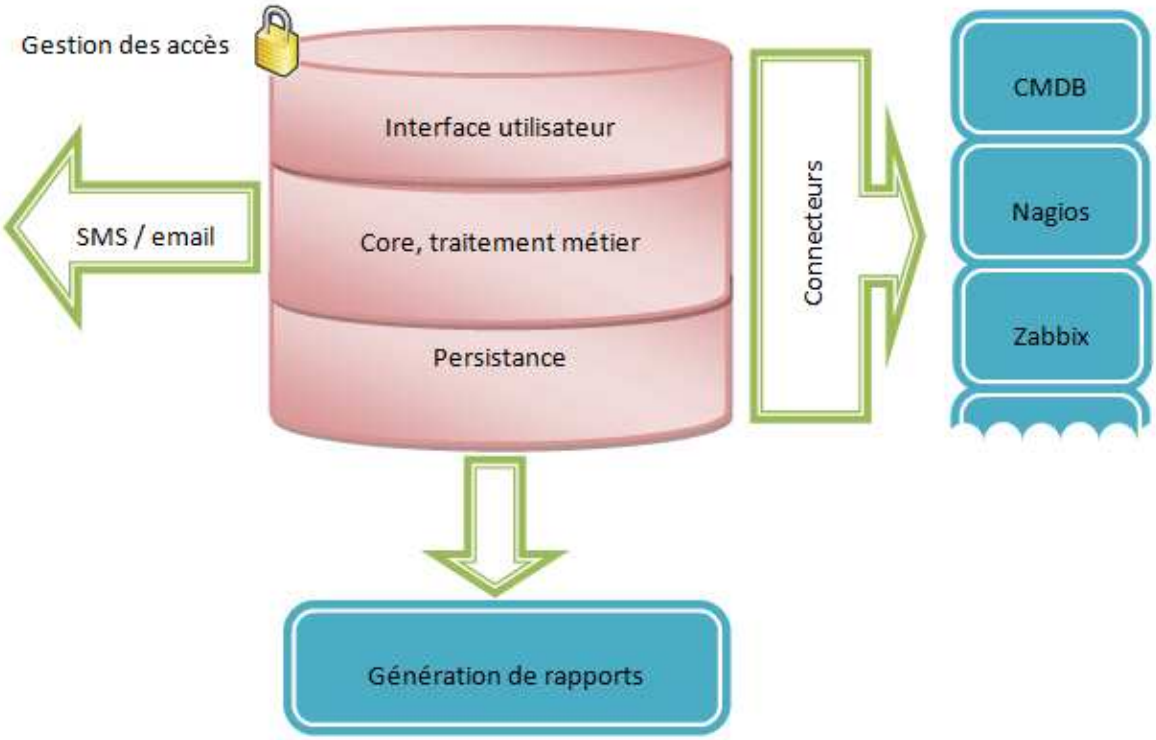

Figure n°10 - Architecture d'une méta-surveillance

3.2.1 L'interface utilisateur, un ensemble de tableaux de bord

Un utilisateur peut consulter un ensemble de tableaux de bord qui le concerne. C'est-à-dire que si l'utilisateur a accès à son application métier, alors il doit pouvoir consulter le statut de la disponibilité de son application. Cela peut se faire de manière automatique, en interfaçant la couche tableaux de bord avec le référentiel de gestion des accès. Les données d'un tableau de bord consistent en un ensemble d'indicateurs et de métriques sur la disponibilité (cf. la section *3.4 - Les tableaux de bord*)

L'interface d'une méta-surveillance ne peut qu'être une interface web. À ce jour, le web est arrivé à sa pleine maturité. Il est impensable de développer une solution en client riche pour le projet qui nous intéresse. Un utilisateur doit pouvoir se connecter en tout

en temps et en tout lieu, sans avoir besoin d'installer de client. Le web est la seule solution qui apporte autant de confort et de souplesse. À ce jour, il est possible d'y développer des interfaces complexes, incluant notamment des menus contextuels, ou la gestion multi-événements, alors qu'il y a à peine quelques années, ceci n'était réservé qu'aux applications en client riche à installer sur chaque poste utilisateur.

Les écrans des smartphones[15] sont maintenant de 4 pouces et permettent aisément d'avoir une interface web qui soit agréable et utilisable depuis n'importe où. Il est tout à fait concevable de développer des tableaux de bord spécifiques pour le format smartphone.

Dans le cadre d'une méta-surveillance, l'utilisateur navigue dans ses tableaux de bord en partant d'une vue globale, par application ou par typologie de service. On peut imaginer une structure par département ou par service d'états-majors.

3.2.2 Un niveau « Core[16] », pour la collecte et le traitement des données

Le niveau « Core » est la partie centrale et la partie la plus complexe d'une architecture de surveillance. Elle permet de collecter les données, les traiter, les filtrer et les corréler entre elles. Le Core offre également un ensemble de connecteurs pouvant aller chercher l'information dans d'autres systèmes, comme une CMDB ou un autre système de surveillance, ainsi que des APIs pour que d'autres systèmes de surveillance puissent venir y écrire des données.

Cette partie doit retenir la plus grande attention lors de la décision d'achat d'un logiciel de surveillance, car si les tests et les connecteurs ne sont pas compatibles avec l'infrastructure existante, alors le projet est un échec.

[15] Smartphone : téléphone complet, qui inclut un agenda, une gestion évoluée de carnets d'adresses, l'accès au Web et aux emails. La plupart des smartphones embarquent un GPS.

[16] Core : se traduit par « cœur » en français. C'est la partie centrale du système.

Plusieurs types de tests peuvent être effectués pour contrôler le bon fonctionnement d'une application :

- Test technique de composant avec un résultat binaire (ex. : ping, présence d'un fichier dans un dossier)
- Test technique avec un seuil (ex : contrôle du nombre d'entrées dans un serveur LDAP, taille disque restante)
- Test end-user
- Test de performance end-user

Les tests « end-user » sont généralement faits depuis un robot de surveillance placé dans le bâtiment ou même dans les bureaux des utilisateurs. Un test end-user consiste à simuler l'activité de l'utilisateur derrière son écran. Le gros avantage de ce genre de test est de simuler et de constater exactement ce que voit l'utilisateur. Si un élément informatique, spécifique à la localisation géographique des utilisateurs, avait un problème, comme un problème sur un élément réseau, cela se verrait avec un test end-user, alors que chaque élément surveillé séparément depuis la salle des serveurs ne donnerait probablement aucune indication.

Les tests de performance consistent à exécuter un certain nombre de requêtes, toujours les mêmes, et à calculer le temps de réponse de chaque requête. Le temps de réponse est comparé à un seuil qui définit si la performance est jugée satisfaisante.

3.2.3 Un niveau de persistance, pour stocker les données

Le stockage des données permet de générer des rapports avec l'historique de la disponibilité d'un système ou de chaque composant. Il est important de bien définir ce qui doit être conservé, afin de garder l'essentiel des données, sans stocker les informations superflues.

On peut garder durant un cours instant les données pour faire du temps réel. Mais cette manière de faire ne permet pas de consulter un historique après coup.

Garder l'historique de l'ensemble des données permet de produire des rapports avec une variation dans le temps. Il faut exécuter des mesures régulières et les conserver. Le stockage de ces données va occuper la base de données et la taille des données augmentera avec le temps, plus ou moins rapidement. Il y a deux manières de stocker les données, soit en conservant l'ensemble des résultats des mesures, soit en ne conservant que les changements d'état. Avec la première manière de faire, les données vont rapidement occuper une taille importante. En ne conservant que les changements d'état, la place occupée sera moindre. À titre de comparaison, si un test est réalisé toutes les 5 minutes, 8640 lignes seront stockées chaque mois. Alors que pour un composant qui n'aura pas eu de changement de statut, il y aura juste l'enregistrement du statut initial, donc 1 seule ligne.

Deux types d'informations sont stockés dans la base, les résultats binaires et les résultats des tests de performance. Un résultat binaire est simple, soit le test est réussi, soit il a échoué. Ce genre de données ne prend pas de place. Par contre, conserver les résultats des tests de performance va inévitablement occuper une part importante de la base de données. Le fait de définir les besoins et les objectifs des rapports avant de prévoir la manière dont seront pérennisées les données, permet de limiter grandement la taille de la base de données.

À titre d'exemple, s'il est nécessaire de faire un graphique sur l'évolution régulière de la performance d'un composant, sur une journée, il est possible de stocker les valeurs capturées toutes les 5 minutes. Par contre, si l'on veut voir l'évolution sur une année, on ne garde que la valeur la plus basse, la plus haute, et une valeur médiane de chaque journée. Cela suffit à présenter un graphique complet, avec un affichage de la variance.

La problématique du stockage des données est à prendre au sérieux pour un système qui lance des milliers de tests chaque heure.

3.2.4 Le référentiel de gestion des accès

Le référentiel de gestion des accès, s'il existe, doit être suffisamment ouvert et complet pour permettre la réutilisation des droits d'accès. Cela permettra de filtrer les accès aux

tableaux de bord de la surveillance des applications. Si l'objectif est de surveiller des applications qui sont déjà sécurisées, pourquoi ne pas s'intégrer à un système de gestion des droits existants ? Cela ajouterait une cohérence à la gestion des accès, et surtout un gain de temps important dès lors que le système d'information dépasse la centaine d'applications. À l'échelle d'une organisation comme celle de l'État de Genève, qui compte près de 25'000 utilisateurs, il n'est pas concevable de gérer les accès aux tableaux de bord ouverts aux utilisateurs d'une autre manière qu'en s'interfaçant à une gestion des accès déjà existante.

Il existe de nombreuses manières de définir une structure des droits, différents modèles de gestion des droits existent. Nous nous appuierons sur une structure simple avec des rôles rattachés à une application. Chaque rôle applicatif correspond à un profil d'accès, donc à un acteur d'un diagramme de cas d'utilisation en UML[17]. Chaque application possède un rôle « utilisateur » qui permet de lancer l'application et d'avoir un minimum de droits. C'est le rôle utilisateur de l'application surveillée qui permet d'accéder au statut de l'application en temps réel dans le tableau de bord utilisateur de la méta-surveillance (cf. la section ci-dessous*3.3 ci-dessous - Les rôles*).

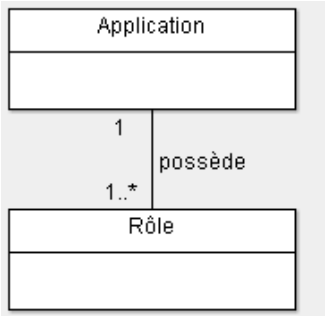

Figure n°11 - Diagramme de classes en UML définissant une application et ses rôles

[17] UML : Unified Modeling Language. Langage de modélisation graphique à base de pictogrammes. Il permet de décrire le système informatique. Le « diagramme des cas d'utilisation » (*use-cases en anglais*) permet d'identifier les possibilités d'interaction entre le système et les acteurs extérieurs au système, c'est-à-dire toutes les fonctionnalités que doit fournir le système.

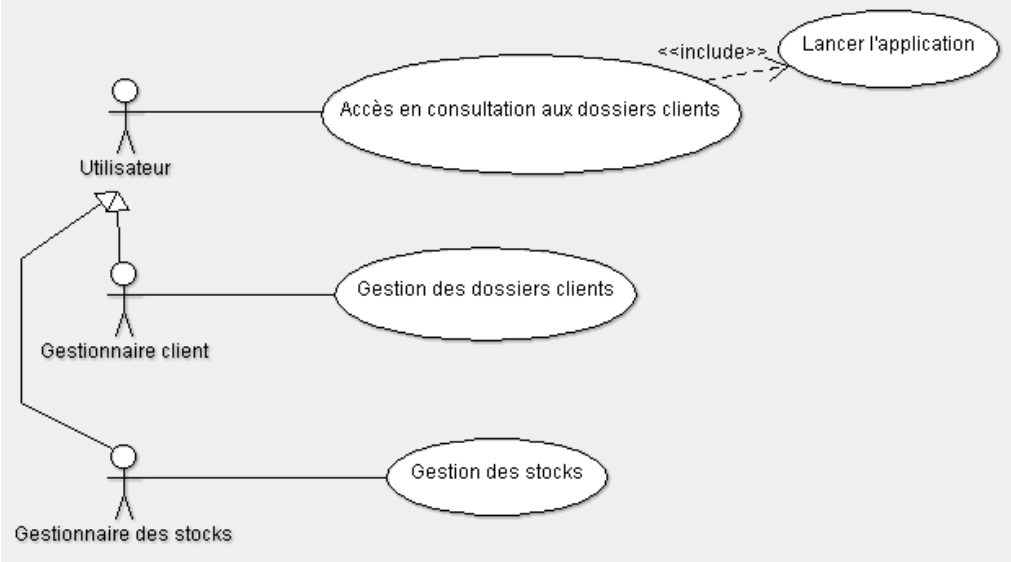

Figure n°12 - Exemple de diagramme de cas d'utilisation pour une application de gestion des clients et des stocks

S'appuyer sur des rôles permet d'avoir une solution indépendante de l'organisation. La définition des rôles est stable dans le temps. Cela ne change qu'avec l'évolution de l'application dans le cas où elle offre de nouvelles fonctionnalités pour de nouveaux acteurs. Pour une application donnée, la liste des acteurs devrait être figée, quelles que soient la langue ou l'organisation dans laquelle l'application sera installée.

Les rôles sont donnés à des services de l'organisation. Les accès donnés de manière nominative doivent être évités au maximum. Ceci est une erreur commune due à une méconnaissance des principes de gestion des accès ou à tentative maladroite de simplification. Lorsque les accès d'une personne sont donnés nominativement, il est nécessaire de mettre à jour chacun de ses accès lors du changement de fonction de ce collaborateur. Afin d'illustrer cela, un responsable de service dans le monde bancaire peut avoir de nombreux accès : gestion des clients (création, modification), gestion des comptes (création, modification, blocage, suppression), gestion des collaborateurs dans le système des ressources humaines (consultation), gestion des comptes rendus d'activité (consultation, validation, et gestion de son propre compte), gestion des investissements et budgets du service (consultation, mise à jour, impression). Dans cet exemple pourtant

basique, Monsieur Piguet a environ 20 rôles différents. En réalité, dans une organisation comme l'État de Genève, un collaborateur possède entre 100 et 200 rôles. Cela implique que si tous les accès sont gérés nominativement, lorsque cette personne doit être remplacée, il va falloir passer en revue la totalité de 100 à 200 rôles pour lui supprimer ses accès, ainsi que rajouter ces 100 à 200 rôles à son remplaçant. Cette tâche n'est souvent pas correctement réalisée, car il y a toujours des oublis, et dans tous les cas il faut compter plusieurs jours, voire dizaines de jours pour que l'ensemble des accès soient traités.

Dans le cas d'une gestion réfléchie par rôle, on ne donne plus des droits à « Monsieur Durand », mais à un rôle (ou profil) de l'organisation. C'est-à-dire que c'est au « Chef de service de l'organisation X » que l'on va donner des droits. Si ce chef de service est amené à être remplacé, il suffit de l'enlever de son unique rôle « Chef de service de l'organisation X » et il perd tous ces accès. De même, son remplaçant sera ajouté dans cet unique rôle. Cela lui confère immédiatement l'ensemble des accès aux services applicatifs dont il a besoin et qui correspondent à sa fonction dans l'organisation.

La mise en place d'une gestion des accès cohérente et intelligente est fastidieuse et nécessite un investissement non négligeable en termes de moyen. Il a fallu plusieurs années à l'État de Genève pour mettre en place une gouvernance de qualité de la gestion des accès à son système d'information. Par contre, le retour sur investissement est important. À moyen terme, une organisation qui a mis ce type de gestion des accès en place ne peut plus s'en passer.

Cela impacte notamment les organes de surveillance et d'audit qui peuvent instantanément contrôler chaque accès et chaque gestion des profils.

Cette gestion des accès définit l'ensemble des rôles qui donnent les droits sur les vues auxquelles aura accès l'utilisateur connecté au système de surveillance. Dans le POC du prochain chapitre, la gestion des accès s'appuie sur le composant du socle sécurité de l'État de Genève, nommé GINA (Gouvernance des Identités Numériques et des Accès), qui référence et gère la sécurité d'environ 650 applications. Ce composant est développé

en Java, mais il existe de nombreuses API[18] qui offrent toujours une solution pour y accéder.

Ce qu'il faut retenir de cette énumération, c'est que chaque application de l'État de Genève peut interfacer sa sécurité avec ce composant transversal. Dans l'outil de méta-surveillance, si chaque application surveillée est intégrée avec le socle sécurité, alors le temps de gestion des accès aux tableaux de bord est nul.

3.2.5 Le référentiel de services applicatifs et des composants

Le référentiel des applications doit référencer la totalité des applications et des composants dont elles dépendent. Dans la norme ITIL, cela s'appelle une CMDB (CMDB dans ITIL v2, et SKMS[19] dans ITIL V3) et cela est rattaché au Centre des Services. L'objectif d'une CMDB est de connaître le parc applicatif dans son ensemble et en détail. Si la mémoire d'un serveur dysfonctionne, on doit pouvoir savoir grâce à la CMDB quels sont précisément les utilisateurs à prévenir, en passant par chaque étape qui relie la mémoire physique d'un serveur aux utilisateurs. Mettre un projet de CMDB en place dans une grande organisation est une tâche laborieuse, qui peut s'étendre sur plusieurs années. Lorsqu'une CMDB est opérationnelle, de nombreux projets peuvent en retirer des bénéfices, dont la surveillance des applications. Il existe deux principales méthodes pour structurer une CMDB, le « référentiel unique » et le « référentiel fédéré ».

3.2.5.1 *Référentiel unique*

Un référentiel unique contient l'ensemble des données, sans faire de référence sur d'autres bases de données. Cette solution permet d'être indépendant, mais oblige chaque acteur à aller documenter et mettre à jour cette base unique. C'est une solution commode si aucun autre référentiel n'existe au préalable dans l'organisation. Néanmoins, s'appuyer sur un tel système devient rapidement

[18] API : (en anglais *Application Programming Interface*) est une interface fournie par un programme informatique. Elle permet l'interaction des programmes les uns avec les autres. C'est une sorte de bibliothèque avec une liste de méthodes que le programmeur peut utiliser.

[19] SKMS : Service Knowledge Management System. Évolution pour ITILv3 de la CMDB de ITILv2.

coûteux, notamment en temps, lorsque différents services sont déjà organisés avec un référentiel qui répond à leur besoin.

3.2.5.2 *Référentiel fédéré*

L'autre manière de constituer une CMDB est de se baser sur une architecture répartie. Pour se faire, on stocke un lien vers d'autres référentiels et on effectue l'agréation de ces données, un peu comme un méta-référentiel. L'avantage de cette approche est que chaque service continu à utiliser ses applications de gestion qui existent déjà, et le référentiel fédéré va simplement faire un lien sur d'autres données contenues dans d'autres référentiels.

Un référentiel consolidé coûte cher en mise en place et en maintenance.

La CMDB peut alors retrouver les utilisateurs utilisant une application donnée. On peut donc imaginer que l'on puisse avoir la liste des utilisateurs impactés par une carte réseau défectueuse sur un serveur.

La gestion des services applicatifs permet de référencer l'ensemble des composants mis bout à bout pour qu'une application fonctionne. Par exemple, pour une application web, il faut le composant sécurité, l'application et une base de données. Le composant sécurité est lui-même dépendant d'un apache, d'un serveur Tomcat, d'un serveur Radius, d'un annuaire Ldap, et d'un provider de sécurité. Le provider de sécurité est lui-même dépendant d'un serveur applicatif Jonas, qui tourne sur un serveur virtuel Solaris 10 qui lui-même tourne sur un serveur physique. Il est possible de descendre dans le détail jusqu'au niveau des composants matériels comme la mémoire ou le processeur.

Comme souvent plusieurs applications dépendent d'un même composant, et il est important de pouvoir définir des composants transversaux : la sécurité par exemple. Le modèle de méta-surveillance doit, en plus de tout cela, permettre de choisir le niveau de granularité sans imposer de descendre trop bas dans le niveau de définition des contraintes d'un serveur. Et même s'il est décidé de descendre dans les couches basses, la gestion ne doit pas s'en trouver plus alourdie.

Avant d'en arriver là, il est nécessaire que chaque service dispose déjà d'un référentiel qu'il maintient à jour régulièrement. La CMDB doit avoir mis en place les connecteurs nécessaires pour faire le lien entre tous ces référentiels. Chaque référentiel spécifique ne peut plus faire évoluer son modèle de données sans en informer la CMDB, afin de pouvoir continuer à se synchroniser avec la CMDB. Une CMDB fédérée est une opération plus complexe à mettre en place qu'un référentiel unique et plus difficile à faire évoluer. Une fois en place, cette solution offre une meilleure souplesse d'utilisation.

3.2.6 Le référentiel des tests

De nombreux logiciels de tests existent sur le marché. Que cela soit des logiciels open source ou des logiciels propriétaires du marché. Certaines suites logicielles sont dédiées à la surveillance de composants, alors que d'autres sont spécialisées dans les tests applicatifs, simulant le comportement d'un utilisateur. Certains éditeurs logiciels proposent leur propre surveillance spécialisée dans les tests de leurs logiciels. C'est notamment le cas de SAP[20], Oracle[21] ou encore Windev[22].

Il existe de nombreux logiciels dont les fonctionnalités se ressemblent beaucoup et qui sont un peu « généralistes ». Leur différentiation réside principalement dans les possibilités de définition de rapports spécifiques ou métiers, ou encore dans la définition des tableaux de bord de statistique. La majorité de ces logiciels proposent de faire des découvertes automatiques du parc informatique.

Il est souvent possible d'exécuter des tests depuis un scanner centralisé, mais le plus souvent il faut installer un agent sur le serveur. Un agent est un exécutable qui tourne sous forme de service sur le serveur distant. Un ordonnanceur déclenche le test à intervalle régulier. Tous les logiciels de test ne sont pas nécessairement compatibles avec l'ensemble des systèmes d'exploitation.

[20] SAP : le leader mondial des PGI (Progiciel de gestion intégré) en entreprise.

[21] Oracle : leur outil de surveillance s'appelle Oracle Enterprise Manager Grid Control.

[22] Windev : est un environnement de développement rapide.

Les logiciels de test, ou robots de surveillance, contrôlent principalement la disponibilité d'un serveur, l'accès à Internet, utilisent des scripts pour connaître la température du processus, de la mémoire, ou font des tests réseaux. Cela permet notamment de détecter bien avant que l'utilisateur ne soit impacté toutes les défaillances matérielles ou lorsque l'espace disque arrive à saturation. De nombreux scripts peuvent être développés, soit dans un langage propriétaire du robot de test, ou dans les langages plus courants comme le Python, le Perl ou même PHP.

À la détection d'un événement qui dépasse un seuil prédéfini, cela déclenche une alerte qui peut être :

- Un simple bip ou une boîte de dialogue d'information sur le serveur
- Un email envoyé à l'administrateur du serveur
- Un SMS envoyé à une liste de numéros préenregistrés
- Un message, JMX[23] par exemple, envoyé à une autre application
- Le lancement d'une procédure de test de diagnostic plus complet
- Le lancement d'un programme tiers

Même si une alerte « error » a été envoyée, le robot de surveillance continue à faire son test à intervalle régulier. À moins que cela ait été spécifiquement défini, aucune nouvelle alerte n'est envoyée si le test est toujours en échec. Par contre, lorsque le test donne à nouveau un résultat valide, alors une alerte « OK » est envoyée pour informer les ingénieurs de l'état de disponibilité de la ressource.

Une solution unique ne couvrant en général jamais l'ensemble d'un besoin, il est courant que plusieurs solutions soient mises en commun pour couvrir l'ensemble du besoin. Dans une solution mixte, s'il existe plusieurs moteurs de surveillance qui gère chacun leur stockage des données, l'interface est normalement commune. C'est sur cette base que je propose une solution de modèle de méta-surveillance.

[23] JMX : Java Management eXtenssions. Service de supervision et d'administration d'applications Java qui s'inspire de SNMP. Il permet de déclencher des actions sur un serveur à distance.

3.3 Les rôles

On distingue principalement trois rôles différents qui agissent dans le cycle de vie d'une donnée circulant dans le SI :

- Le propriétaire de la donnée
- Le gestionnaire de la donnée
- L'utilisateur ou le consommateur de la donnée

On retrouve ce même principe dans la surveillance d'un système d'information. Le propriétaire est représenté par le service de pilotage qui est responsable de la surveillance. Le gestionnaire est un ingénieur système qui est responsable d'un serveur ou un intégrateur qui s'occupe d'installer et de garantir que l'application et ses dépendances fonctionnent. Quant à l'utilisateur, c'est toujours le même rôle, à savoir celui pour qui l'application est destinée.

Dans les cas du modèle développé dans le cadre de ce mémoire, la complexité est un peu plus étendue, et 7 rôles ont été définis. Il est évidemment possible d'aller beaucoup plus loin dans la granularité, mais ce n'est pas le but recherché. Il s'agit simplement de définir les profils principaux. Certains ont été doublés, comme le rôle « concepteur », afin de séparer le monde composant du monde applicatif. De même que pour le rôle utilisateur, on fait la différence entre un utilisateur d'une application et un chef de service ou un directeur qui, même sans être utilisateur de l'application, souhaite avoir une vue d'ensemble de la disponibilité de ses serveurs.

3.3.1 Les cas d'utilisation

Les cas d'utilisation en UML décrivent les interactions entre les acteurs et le système. Les acteurs sont rattachés à une fonctionnalité de l'application. Il existe un héritage entre tous les rôles et le rôle utilisateur, notamment pour avoir le droit de lancer l'application.

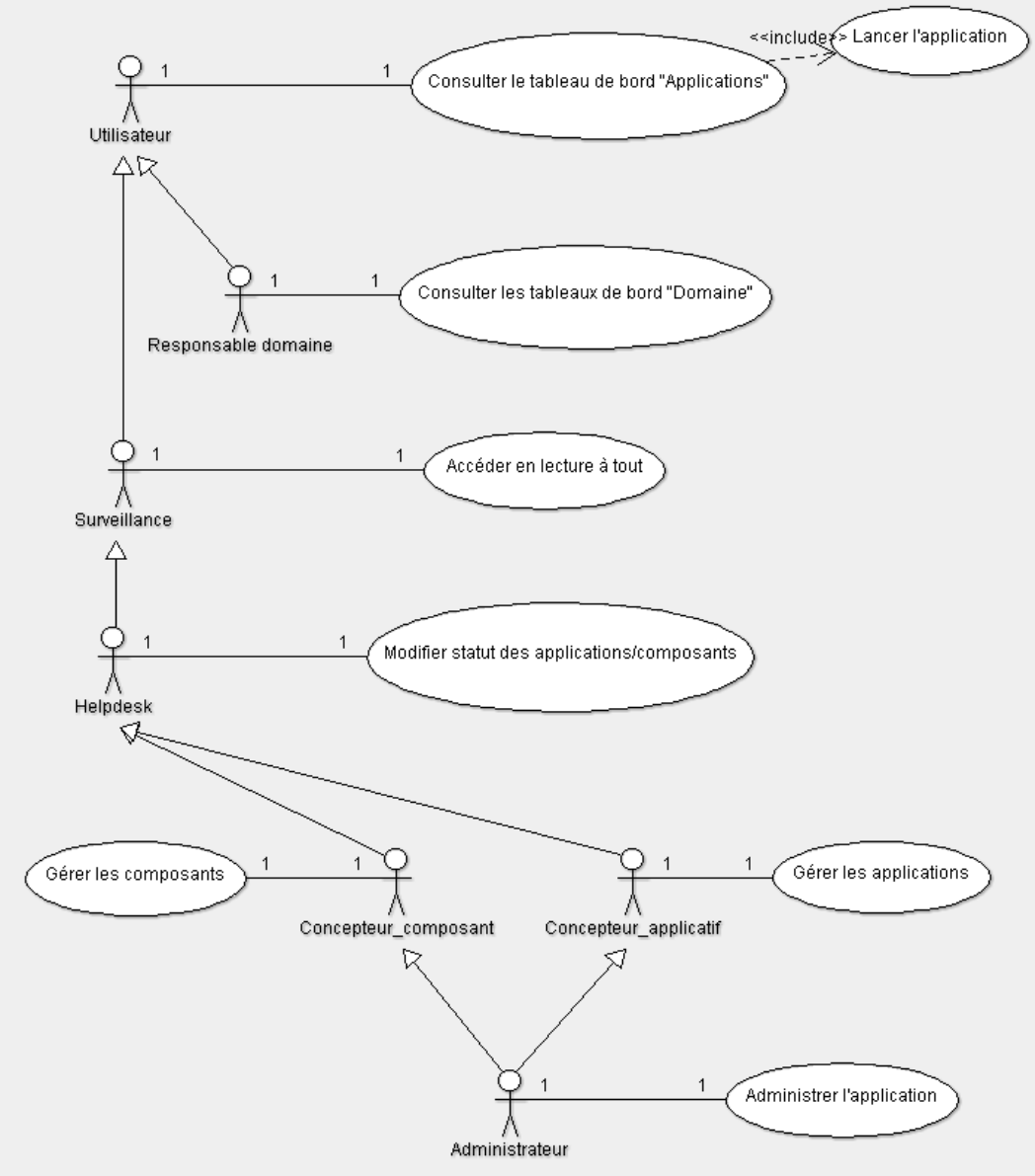

Figure n°13 - Diagramme de cas d'utilisation de la méta-surveillance

3.3.2 Description des rôles de la méta-surveillance

3.3.2.1 *Utilisateur*

Un utilisateur peut accéder aux tableaux de bord et consulter la disponibilité des applications dont il est utilisateur afin de savoir si ses applications sont toutes

disponibles. On peut imaginer qu'un composant transversal impacte plusieurs applications et perturbe ainsi grandement le travail du collaborateur. C'est pourquoi il est nécessaire d'offrir une vue complète à l'utilisateur afin que celui-ci puisse éventuellement consulter quelles applications ne sont pas impactées par ce problème et fonctionnent encore. De cette manière, il est à même de faire une autre tâche le temps de résolution de la panne afin de ne pas être bloqué dans son travail.

Un utilisateur n'a aucun moyen de contrôler le détail d'un composant d'une application ni d'interagir avec le système, il n'a qu'une vue de consultation applicative.

Tous les autres rôles héritent du rôle utilisateur.

3.3.2.2 *Responsable de domaine applicatif*

Un responsable de domaine applicatif peut interroger toutes les applications dont il est responsable au niveau d'un domaine. Un domaine correspond à un ensemble d'applications appartenant à une même « catégorie métier ». Cela peut être le cas pour un responsable d'un service, un directeur ou un responsable métier qui souhaite vérifier la disponibilité des applications qui sont utilisées par ses collaborateurs. Il peut consulter la disponibilité de l'ensemble des applications des services métiers dont il est responsable. En plus des applications qu'il voit avec le rôle hérité Utilisateur, il voit toutes les applications du domaine, même s'il n'a pas les droits d'accès à l'application métier.

3.3.2.3 *Surveillant*

Le surveillant a accès, en lecture uniquement, à l'ensemble des applications et des composants, dans le détail. Il peut choisir de générer un incident sur la base d'une alerte. Le rôle de surveillant est dédié au service Pilotage qui assure la disponibilité des applications en production.

3.3.2.4 *Centre de Service (Helpdesk)*

Il arrive qu'un incident soit détecté par l'utilisateur avant que le système de surveillance ne le détecte. Soit parce que le cas de test n'a pas été pensé, soit parce que le système fait un test à une fréquence qui a permis à l'utilisateur d'être plus rapide. Dans ce cas, il est nécessaire que le Centre de Service puisse lui-même changer le statut d'un composant ou une application et le mettre en erreur. Lorsque le statut est mis à jour manuellement, il ne faut pas que le système de contrôle automatique puisse rechanger le statut. Un statut mis à jour manuellement en erreur, doit être remis à jour manuellement en « OK », soit par le Centre de Service, soit par le surveillant, soit par le concepteur responsable du composant ou de l'application.

Dans un cas idéal, ce n'est pas manuellement que le statut devrait être mis à jour, mais cela se fait automatiquement à l'ouverture d'un incident. Cela implique que le système de surveillance doit être interfacé avec l'outil de gestion des incidents. À l'ouverture du ticket, il doit être possible de référencer le composant ou l'application, et son statut mis à jour automatiquement. À la fermeture de l'incident, les tests de l'élément concerné sont lancés automatiquement, afin de remettre à jour son statut.

3.3.2.5 *Concepteur applicatif*

Le concepteur applicatif a la charge de définir les tests applicatifs et développer les scénarios end-user. Il doit également définir la dépendance entre les applications et les composants. Ceci se fait en relation avec le CMDB si elle existe.

3.3.2.6 *Concepteur composant*

Le concepteur composant est un ingénieur responsable d'un type de composant technique ou transversal. Il a une vue supplémentaire regroupant l'ensemble des composants techniques dont il est responsable. Au même titre que le concepteur applicatif, celui-ci définit les scripts de test pour le type de composant dont il est responsable. En règle générale, une fois qu'un script est fonctionnel, il peut être

appliqué à tous les composants du même type.

Toujours en lien avec la CMDB, il doit définir les dépendances entre les composants et sous-composants, ou faire les regroupements des services en cluster. Il peut à tout moment remonter sur une vue listant les applications qui dépendent d'un composant.

3.3.2.7 *Administrateur*

Un administrateur accède en écriture à l'ensemble des écrans. Il est notamment responsable de définir les profils de gestion des droits.

3.3.3 Description des cas d'utilisation

3.3.3.1 *Lancer l'application*

Lancer l'application signifie que nous avons les droits minimums pour être reconnu et que l'application s'ouvre. Le détail de ce qui va s'afficher ensuite va dépendre des autres droits que possède l'utilisateur.

3.3.3.2 *Consulter le tableau de bord « Applications »*

Un utilisateur peut consulter la liste des applications dont il est utilisateur selon le référentiel de gestion des accès. Il voit également le temps depuis lequel une application est dans cet état.

3.3.3.3 *Consulter les tableaux de bord « Domaine »*

Le tableau de bord d'un domaine regroupe l'ensemble des applications qui appartiennent à un domaine métier spécifique. Il consulte l'état du domaine en temps réel, et peut contrôler l'historique des changements d'état de disponibilité. Il voit également le temps depuis lequel un domaine est dans cet état.

Depuis chaque domaine dont il est responsable, il peut consulter la liste des applications du domaine. Il s'agit de la même vue qu'un utilisateur.

3.3.3.4 *Accéder en lecture à tout*

Tout voir, c'est-à-dire la liste des domaines, la liste des applications et l'ensemble des composants dont elles dépendent. Accès à tous les tableaux de bord, critiques, domaine, statut et composant.

3.3.3.5 *Modifier le statut des applications/composants*

Modifier le statut d'un composant, y ajouter un commentaire

Modifier le statut d'une application, y ajouter un commentaire.

3.3.3.6 *Gérer les applications*

Créer, modifier, supprimer une application.

Définir les cas de test pour une application.

Définir de quels composants une application dépend en relation avec la CMDB.

Définir les cas d'utilisation critiques de l'application.

3.3.3.7 *Gérer les composants*

Créer, modifier, supprimer un composant.

Définir les cas de test pour un composant.

Définir de quels sous-composants un composant dépend.

Définir les cas de test critiques pour un composant.

3.3.3.8 *Administrer l'application*

Créer, modifier et supprimer les profils d'accès et les liaisons avec le référentiel de gestion des accès.

Créer, modifier, supprimer un tableau de bord.

3.4 Les tableaux de bord

La mise en place de tableau de bord doit répondre à plusieurs objectifs :

- Avoir une vision de tous les composants, savoir si chaque composant répond correctement, que celui-ci soit matériel (processeur, mémoire, disque dur…) ou que ce soit un service applicatif (serveur radius, web service…).
- Avoir une vision end-user[24], c'est ce que voit le client. On peut imaginer qu'un composant tombe, mais si celui-ci est redondant, l'utilisateur ne sera pas impacté, ni même informé.
- D'une vision end-user, permettre à l'utilisateur d'avoir une notion de performance presque en temps réel puisqu'aujourd'hui, le fait qu'une application réponde n'est plus suffisant, il faut aussi qu'elle réponde dans un temps donné. Il est possible que si une application subit des lenteurs, cela puisse venir du serveur ou d'un composant transversal. Dans le cas d'un composant transversal, cela impactera donc plusieurs applications, d'où l'importance de la mise en place d'une méta-surveillance.

Les applications pouvant être ouvertes également sur Internet ou sur des réseaux spécifiques, il est nécessaire à ce moment de pouvoir indiquer si l'application répond correctement depuis tous les points d'où elle est accessible.

La mise en place de ces tableaux de bord permettra d'avoir une visibilité en temps réel du fonctionnement de l'ensemble du système d'information. Cette information peut offrir une vision :

- par service métier
- par type de composant
- selon le niveau de criticité de l'application

[24] End-user : signifie « d'un point de vue de l'utilisateur ». Dans le cas d'un test, plutôt que de tester chaque composant pour dire s'il est fonctionnel ou non, on va juste faire un test dans les mêmes conditions que l'utilisateur, sur un poste client. Si cela fonctionne, alors cela pourrait signifier que l'ensemble de la chaine fonctionne. Mais il manque le contrôle de savoir si tous les serveurs en cluster fonctionnent. Le test « end-user » reste néanmoins un test fondamental à exécuter.

- selon les accès utilisateur
- par statut

En fonction du rôle, les vues et les actions possibles sur ces vues peuvent être différentes. Selon les vues, il est essentiel d'avoir le détail de ses composants qui s'affiche et de savoir quel composant est redondant ou non. Pouvoir mettre en évidence cette notion permet encore une fois d'anticiper le niveau de criticité d'un composant et de définir le niveau de priorité.

Au niveau des composants, il faut également un écran qui permet de voir l'ensemble des applications dont il dépend, avec la chaine complète, des composants et sous-composants. L'inverse est nécessaire aussi, c'est-à-dire avoir la possibilité de consulter l'ensemble des applications qui dépendent d'un composant.

Sur chaque écran de composant, il doit être possible d'accéder directement à la documentation du composant, soit en direct, soit par référence.

3.4.1 Les différentes vues

3.4.1.1 *Utilisateur*

L'utilisateur accède à un écran avec la liste des applications auxquelles il a droit. Pour chaque application, il voit son nom et son statut. Il est possible d'afficher un ensemble d'attributs complémentaires tels qu'un numéro unique, un descriptif, l'URL de connexion, le temps depuis lequel l'application est dans ce statut. En cas de warning, il est possible d'afficher le détail à savoir le statut technique, métier et de performance. Un utilisateur ne peut pas avoir accès à plus de détails que la liste de ces applications, en effet la liste des composants dont l'application dépend reste toujours cachée pour l'utilisateur. Pour rappel, si un composant est en « warning », cela ne change pas le statut de l'application, car elle reste fonctionnelle pour l'utilisateur.

3.4.1.2 *Responsable de domaine*

Le responsable d'un domaine applicatif accède à un écran avec les domaines desquels il est responsable, et en cliquant sur un domaine, il voit l'ensemble des applications appartenant au domaine, qu'il y ait accès ou non. Pour chaque application, on retrouve les mêmes informations que pour le rôle utilisateur.

3.4.1.3 *Concepteur applicatif et concepteur composant*

Les concepteurs ont plusieurs vues à disposition. Depuis le même écran que le responsable de domaine, ils accèdent à l'ensemble des domaines. Un concepteur a un statut complémentaire qui change si au moins une application, un composant ou sous-composant est dans un statut autre qu'« OK ». Cette vue n'est pas visible par les utilisateurs, mais prend toute sa valeur pour un concepteur. Il peut ainsi rapidement visionner les applications / composants qui posent problème en descendant dans le détail de la vue. Avec une vue en arbre directement ouverte sur la branche du composant qui pose problème, il est rapide et aisé pour un ingénieur d'intervenir.

Un concepteur possède plusieurs écrans qui fonctionnent tous selon le même principe, mais avec des filtres différents :

- Un écran avec la liste des applications / composants dont il est responsable
- Un écran avec la liste des applications / composant en « warning » ou en « error »
- Sur la base d'un composant, un écran avec toutes les applications qui dépendent du composant sélectionné
- Un écran avec la liste des composants dont il est dépendant

Pour chaque composant, il y a le nom du composant, son identifiant, l'adresse IP du serveur où il est installé, sa documentation, et qui en est responsable. De cette manière, il peut aisément prévenir la personne qui n'aurait pas pu être informée.

3.4.1.4 *Administrateur*

Un administrateur a les mêmes vues qu'un concepteur. Un filtre lui permet de définir une responsabilité et de lister ainsi tous les composants attachés à cette responsabilité.

3.4.1.5 *Surveillant*

Le surveillant a les mêmes écrans que les administrateurs, mais en lecture seule.

3.4.1.6 *Centre de Service*

Le Centre de Service a les mêmes droits que le surveillant, à la différence qu'il peut forcer l'état d'une application ou d'un composant en lançant un test manuellement, afin de pouvoir réagir sur un appel d'un utilisateur qui lui annonce la panne d'une application. Il peut y ajouter un commentaire qui sera lié au ticket de panne.

3.4.2 Les indicateurs

Un indicateur est un outil d'évaluation, de pilotage et d'aide à la décision. Sa représentation peut être numérique, graphique ou textuelle. Dans le cas de la surveillance, un indicateur représente l'état de fonctionnement d'une application ou d'un composant à un instant donné. Un indicateur est un résumé d'informations permettant en un coup d'œil d'avoir une idée précise de l'état d'une application ou d'un ensemble d'applications. Le but des indicateurs est de mesurer la disponibilité ainsi que de la performance des applications.

Encore une fois, l'objectif n'est pas de faire un test de performance, mais de récupérer cette information et de trouver le moyen adéquat pour le présenter à l'utilisateur ou encore, de lever une alerte aux responsables des composants ou des applications qui posent problème.

On distingue trois types d'indicateurs en surveillance :

- Les indicateurs d'état
- Les indicateurs de temporelle
- Les indicateurs de performance

Les indicateurs de surveillance de systèmes d'information peuvent contenir des sous-indicateurs ou indicateurs complémentaires. Des indicateurs textuels ou numériques sont ajoutés à un indicateur d'état pour venir compléter l'information. Les plus pertinents sont affichés directement, et le détail des indicateurs peut être affiché après une action de l'utilisateur.

3.4.2.1 *Les indicateurs d'état*

Les indicateurs d'état de surveillance des systèmes d'informations travaillent sur quatre niveaux d'état associés généralement à une couleur, de manière assez logique :

- OK → Vert
- Warning → Jaune
- Error → Rouge
- Unknow → Gris

Chaque niveau peut être accompagné d'un message détaillé, notamment pour indiquer la raison du warning ou de l'erreur.

La dépendance entre les applications et les composants peuvent influencer un indicateur de plus haut niveau. L'affichage d'un warning en jaune pour un composant peut signifier qu'un sous-composant en cluster est en erreur, donc en rouge.

En plus de la notion d'indicateur, il y a une pondération en fonction du rôle qui consulte les indicateurs. Un utilisateur ne verra pas la même chose qu'un ingénieur. Si un serveur en cluster tombe, la disponibilité est assurée pour l'utilisateur, alors que l'ingénieur devra, lui, détecter qu'il y a un problème et réagir vite. Il n'est pas de l'intérêt du centre informatique qui met à disposition les

indicateurs de surveillance du système d'information auprès de ses utilisateurs de prévenir de la moindre panne. Il est donc nécessaire de prévenir l'utilisateur uniquement en cas de panne majeure, ou du moins pour une panne qui a un impact sur le bon fonctionnement du SI. C'est pourquoi il y a un système de pondération, ou de filtre, en fonction du rôle qui observe le tableau de bord.

Un utilisateur ne peut consulter que le tableau de bord qui est lui destiné, alors qu'un ingénieur (le rôle Concepteur) a une vue double. En effet, il est important qu'un ingénieur puisse être alerté pour chaque panne d'un composant, et il doit également avoir la possibilité d'afficher l'écran prévu pour un utilisateur.

3.4.2.2 *Les indicateurs temporels*

La première question qui est posée quand un composant tombe en panne est « Depuis quand ce composant est-il indisponible ? ». À côté de chaque indicateur d'état, il est pratique d'avoir un indicateur temporel qui définisse depuis quand un composant est dans cet état.

Pour un composant, il doit être possible d'aller voir l'historique des changements d'état. Il doit être possible de savoir depuis combien de temps une application est indisponible, ou inversement depuis combien de temps une application fonctionne correctement ? C'est une donnée fondamentale pour contrôler la qualité d'un SLA.

Il existe un risque si un outil de surveillance provoque une erreur sur le tableau de bord, alors que l'application métier fonctionne correctement. Cela pourrait induire en erreur l'utilisateur sur la qualité du service rendu. Inversement, si une application ne fonctionne pas, alors que l'indicateur est vert, cela génère inévitablement des contestations de la part des utilisateurs.

3.4.2.3 *Les indicateurs de performance*

Les indicateurs de performance relèvent de la qualité de service. Souvent les indicateurs de performances sont représentés à l'aide de graphique avec une ligne

de référence, ou un tachygraphe. Un indicateur de performance, encore plus que les autres mesures, dépend considérablement du lieu depuis lequel est prise la mesure. Souvent une application fonctionne bien, mais la performance peut être influencée par de nombreux facteurs, comme la charge réseau ou le poste de travail utilisateur.

Les indicateurs de performance sont dédiés au centre informatique. Il n'est pas prudent de présenter ces graphiques à l'ensemble des utilisateurs. Ces indicateurs permettent de faire de la qualité et d'anticiper d'éventuelles pannes qui pourraient être dues à une surcharge.

3.4.3 Filtre sur les indicateurs en fonction du rôle

3.4.3.1 *Utilisateur*

L'utilisateur ne voit que les applications, il n'a pas de vue sur le détail des composants.

- Un indicateur **vert** signifie que l'application fonctionne et répond correctement, même si un composant en cluster ne fonctionne pas. Cela n'a pas d'impact pour l'utilisateur final. Cela déclenche par contre une alerte pour l'ingénieur responsable de ce composant.
- **Jaune** signifie qu'il y a un problème métier ou partiel, mais que l'application reste fonctionnelle dans son ensemble. Cela peut être le cas d'un problème de performance. Ça peut être le cas lorsque qu'une application fonctionne, mais dont l'impression, ou encore la connexion à une autre application ne fonctionnent pas, alors que la majeure partie des fonctionnalités sont toujours disponibles.
- **Rouge** signifie que l'application est indisponible ou que les performances rendent le système inutilisable.

3.4.3.2 *Responsable de domaine*

Un responsable de domaine a la même vision que le rôle utilisateur, mais pour

l'ensemble du domaine dont il est responsable, même s'il n'est pas utilisateur de l'application. C'est le cas d'un responsable de service ou d'un directeur qui veut s'assurer de la disponibilité des applications mises à disposition de ses collaborateurs.

3.4.3.3 *Surveillant, Centre de Service, concepteur applicatif et composant*

Ces rôles peuvent tout d'abord consulter la vue des applications :

- **Vert** signifie que l'application fonctionne correctement, qu'il n'y a aucun problème détecté nulle part.
- **Jaune** signifie qu'un composant de l'application n'est pas bon. Même s'il s'agit d'un seul sous-composant, cela permet de voir immédiatement où sont les problèmes.
- **Rouge** apparaît lorsque l'application n'est plus utilisable du tout.

Les rôles de surveillant et de concepteur reçoivent une alerte (SMS/email) pour chaque composant ou sous-composant qui passe au rouge.

- **Vert** signifie que le composant offre le service demandé.
- **Jaune** apparaît lorsqu'un des composants redondants a un problème sur un des clusters (donc le sous-composant défectueux est rouge). Le rôle de surveillance peut consulter le détail de tous les composants et sous-composants.
- **Rouge** s'affiche lorsqu'un composant est complètement hors service. Cela peut être le cas d'un composant en cluster où tous les clusters sont tombés. Ou alors à la consultation du détail d'un sous-composant qui est lui-même en panne.

3.4.4 Réflexion sur la pondération d'indicateur

Si on parle d'indicateur, il est également nécessaire de définir si un service en jaune est plus important qu'un autre service en jaune. Établir une pondération sur les indicateurs est le seul moyen de gérer des priorités d'intervention entre deux composants. On

pourrait imaginer mettre une couleur intermédiaire pour le warning, comme mettre un indicateur orange, ou avoir une valeur numérique de criticité. C'est-à-dire que si 50 % du service est assuré, alors l'indicateur reste jaune. Mais si on passe à un niveau d'erreur plus grand que 50 %, alors l'indicateur pourrait être orange, pour augmenter le niveau d'alerte et permettre de gérer les priorités.

Pour pousser la réflexion un peu plus loin, il serait intéressant d'envisager la possibilité d'augmenter la pondération d'un indicateur lorsque celui-ci impacte de nombreuses applications. En effet, si plusieurs pannes se produisent en même temps, il est probable que l'ingénieur soit contraint de faire un choix sur la panne à considérer en premier.

Voici quelques exemples de pondérations qui pourraient être mises en perspective :

- Le nombre d'applications finales qui dépendent de ce composant.
- Le nombre d'utilisateurs finaux qui dépendent de ce composant.
- Le nombre d'applications utilisées dans le cas d'un guichet, avec un service à la clientèle
- Une pondération de la criticité de l'application. C'est une valeur définie manuellement, estimée selon des conventions avec les clients.
- Une pondération sur le coût horaire d'une panne pour l'application. Cela permet de définir le composant le plus « cher ».
- Le nombre d'instances redondantes du composant. Si un composant est fortement redondant, alors il n'est plus critique.
- L'application dont le SLA définit la durée d'intervention la plus courte.
- L'application surveillée contrôle ou commande d'autres services, comme un batch ou un ordonnanceur.
- Une pondération si c'est une application qui peut présenter un risque physique sur des humains (le 117, 118, 144[25], ou alors par exemple sur un contrôle de débordement d'un barrage qui pourrait provoquer des inondations).
- Durant l'établissement des scripts de test GUI, certains cas d'utilisation sont plus critiques que d'autres. Dans un système de ressources humaines, le

[25] Numéro d'urgences en Suisse : 117 = Police, 118 = Pompier et 144 = Urgences médicales

déclenchement du processus de paie est bien plus important que le stockage des photos des collaborateurs. Le responsable de l'application peut ainsi définir que le processus « gestion des paies » est critique. Il est alors nécessaire de définir quels composants sont indispensables à ce processus. Alors que les composants nécessaires au stockage des photos ne sont pas utilisés dans le processus de paie et ne sont donc pas critiques, même si l'application est une application majeure pour le bon fonctionnement de l'entreprise qui l'utilise.

- Le nombre de tickets ouverts sur un incident.

C'est dans la CMDB que la criticité des composants et des processus doit être définie.

Il faut pouvoir définir les dépendances de composants en fonction des cas d'utilisation, ou des processus, et non pas de l'application simplement. Car certains composants n'influencent que certaines parties de l'application et il n'est pas nécessaire de gérer une alerte.

Définir une pondération au niveau des indicateurs peut se révéler crucial dans le cas d'un incident afin de déterminer les priorités d'intervention. Un ingénieur qui s'occupe d'un composant ou d'un serveur n'a souvent pas la notion de l'importance ou de la criticité de l'ensemble du parc dont il s'occupe. Il est aussi probable qu'un ingénieur intervienne en remplacement d'un collègue ou lors d'un service d'astreinte[26], et qu'il ne sache pas si une panne est un serveur de sauvegardes ou si c'est le serveur central qui pourrait paralyser la moitié de l'entreprise.

Il est aussi courant qu'un ingénieur système s'occupe de monter et d'installer des serveurs physiques qui sont mis à disposition des intégrateurs qui, eux, vont installer l'environnement nécessaire, le serveur applicatif et les applications. L'ingénieur système ne sait même pas quelles applications pourraient tourner sur son serveur physique. Si la carte réseau provoque des problèmes de connexion, il ne peut pas deviner le nombre d'utilisateurs impactés, ni aucun des autres indicateurs définis plus haut.

[26] Service d'astreinte : signifie que l'entreprise met en place un système où une personne peut être atteignable 24/24h, 7/7j pour intervenir en cas de problème.

Tous ces indicateurs permettent pourtant de définir si le SLA est respecté ou non. Si le SLA définit un accord de 96 % de disponibilité entre 7h et 20h, le fait de respecter ces 96 % de disponibilité signifie que le SLA est respecté à 100 %. Dans le cas où un SLA n'est pas respecté, à savoir si seulement 95 % du SLA est couvert, alors il est nécessaire de vérifier si l'infrastructure correspond vraiment au besoin émis par le client.

À l'inverse, si un SLA est respecté à 103 %, cela signifie qu'on fait de la sur-qualité et cela coûte évidemment au client. Ce dernier ne voudra pas payer une infrastructure qui coûte cher alors que son besoin n'est pas aussi exigeant. Les coûts d'infrastructure peuvent être diminués. L'aspect stratégique de la surveillance d'applications est une solution qui permet aussi de vérifier la cohérence des investissements de haute disponibilité.

3.5 Le temps réel

La mise à jour d'un indicateur en temps réel signifie qu'il est mis à jour exactement en même temps que le changement d'état du composant surveillé. Dans notre cas, la notion de temps réel est légèrement différente, car il n'est que rarement envisageable qu'un système lance lui-même une alerte en cas de panne, puisque justement il est en panne. Par contre, il est courant qu'un composant dépendant de ce système lance rapidement une alerte. Ou du moins, un composant génère un log pour signifier que tel ou tel composant est indisponible. Dans la mesure du possible, il va falloir s'appuyer sur ce genre de détections qui peuvent être complémentaires à un robot qui lui lance des scripts à intervalles réguliers. Cela se fait en inspectant les logs, ou lorsqu'un composant tiers n'arrive pas à se connecter, celui-ci peut générer une alerte directement à travers une API de surveillance.

Malgré tout, il n'est pas toujours envisageable de le faire et encore moins de le corréler à une surveillance globale. La solution passe donc par un système auxiliaire et indépendant qui lance un test à intervalles réguliers. Il n'est pas raisonnable de faire un test chaque seconde de l'état de chaque composant d'un système pour des raisons de charge et de stockage des données. En fonction des tests et de la criticité du composant, ceux-ci pourront être effectués toutes les 5 à 15 minutes, ou parfois, un test par heure peut se

révéler suffisant.

Le contrôle d'un batch se lance après que le batch ait fini son opération. Le contrôle se fera sur la cohérence des données. Si le batch est lancé une fois par semaine, il n'est pas nécessaire de lancer un test toutes les heures. Un test unique exécuté après le batch est suffisant.

On constate que la notion de temps réel est parfois utilisée de façon abusive, et en cas d'indisponibilité, il risque d'y avoir un appel de l'utilisateur avant que le système lance une alerte. Par contre cela viendra confirmer la panne, permettra de calculer le SLA et de communiquer avec le client, ainsi que de conserver un historique.

Bien que la mise à jour de ces indicateurs ne soit pas immédiate, on parle néanmoins de temps réel, car elle est automatique et vu l'importance de la surveillance d'un SI complet, nous considérons que le délai de quelques minutes est un délai convenable. La mise à jour des tableaux de bord doit néanmoins pouvoir être rapide après un test lancé manuellement, notamment après la résolution de la panne, cela permet de rassurer les utilisateurs de manière concrète et rapide.

3.6 Système d'alertes

Lorsqu'un indicateur passe au rouge, il faut envoyer une alerte. Les types et seuils d'alertes sont définis spécifiquement pour chaque application ou composant et peuvent être différents en fonction de chacun. Une alerte peut être envoyée par SMS, ou par email.

Les alertes doivent être envoyées quand un problème est détecté, ainsi que l'alerte qui indique le retour à la situation normale.

La plus-value d'un système intégré, c'est qu'une méta-surveillance est capable de détecter quand une alerte est levée à un niveau N de composant, c'est peut-être que le composant N-1 ou N-2 est en panne. Il n'est donc pas nécessaire d'envoyer une alerte pour l'application N, alors que c'est un composant N-2 qui est responsable de la panne. N

dépend simplement de N-2.

Néanmoins, le responsable d'un composant souhaiterait probablement recevoir une information d'indisponibilité de son composant, alors même que c'est un sous-composant qui pose problème. Le risque en filtrant est d'avoir deux problèmes et de penser que si N-2 pose problème alors il n'a pas lieu de penser qu'il peut y avoir un second problème sur N ou N-1 et de le louper.

Une méta-surveillance qui intègre et fait une synthèse des différents systèmes de surveillance, est capable de détecter qu'une application qui déclenche une alerte est référencée avec un composant qui a déjà déclenché une alerte, alors cette information devrait être mise dans le mail qui sera envoyé automatiquement. C'est ensuite la responsabilité de la personne qui traite le ticket de lire l'information au complet et d'avoir le bon comportement.

Lorsqu'une personne reçoit un ticket, le bon réflexe serait de consulter le tableau de bord de surveillance de l'application pour constater d'elle-même si un composant dont dépend l'application ne serait pas déjà en alerte.

À la résolution de l'alerte de niveau N-2, il faut relancer tous les tests des composants qui étaient dans un état différent qu'« OK » et qui dépendaient de ce composant.

On a pu constater que si le répertoire de log d'un serveur proxy est plein, cela peut bloquer le serveur proxy. En réalité, le proxy fonctionne, alors que c'est juste un problème de taille sur le disque dur. Si le proxy ne répond plus, l'application ne fonctionne plus non plus, mais ce n'est pas un problème de l'application.

3.7 Les scénarios possibles pour une méta-surveillance

Sur la base d'un modèle de méta-surveillance, plusieurs scénarios sont envisageables :

- Un développement complet.

- S'appuyer sur un outil open source, comme Nagios[27] ou Zabbix
- Utiliser une solution commerciale
- Un mixte des 3 solutions.

Avant tout, il est important de définir le contexte. Dans notre cas, il s'agit bien d'étudier un scénario plausible et applicable à la surveillance du SI de l'État de Genève. Il est évident que les choix et les critères sont différents s'il s'agit d'une PME possédant une dizaine de serveurs.

Un SI d'une organisation comme l'État de Genève est particulièrement hétérogène. Les serveurs sont répartis sur plusieurs salles machines éloignées géographiquement et qui assurent entres-elles une redondance. Les 25'000 utilisateurs sont localisés sur plus d'une centaine de sites différents aux quatre coins du Canton de Genève.

3.7.1 Développement complet

Cela reviendrait à faire l'intégralité de l'analyse et les développements en interne. L'expérience démontre que lorsque le marché d'une solution n'est pas mature, il peut y avoir d'excellentes idées à développer en interne. Cela demande du temps et coûte évidemment plus cher. Puis, le jour où la solution est enfin exploitable, il est courant que le marché offre alors des solutions commerciales ou open source aussi performantes, voire meilleures, et à moindres frais. À compter du moment où des solutions efficaces existent sur le marché, les développements internes ont une durée de vie de 1 à 2 ans.

Les outils de surveillance arrivent aujourd'hui à un niveau de maturité tel, que ce serait commettre une erreur que de vouloir partir de zéro.

Il n'y a aucun avantage à partir de zéro pour un projet aussi ambitieux que de surveiller l'infrastructure informatique et le système d'information pour l'État de Genève. Cela nécessiterait une force de développement importante, un budget élevé et une volonté de tout réinventer.

[27] Nagios : application open source permettant de faire de la surveillance système et réseau.

3.7.2 Choisir un outil open source ou gratuit

En choisissant une solution open source ou gratuite, le CTI respecterait ainsi une directive du Conseil d'État sur l'orientation des choix des applications vers le logiciel libre. Il s'agit de la « Mesure 28 [28]» qui fait partie du premier plan de mesures[29] édicté par le Conseil d'État le 29 novembre 2006.

Le fait de prendre un outil open source, en général gratuit, n'est pas un argument en soi, car cela implique aussi une mise en service qu'on a souvent tendance à oublier. La mise en place et l'adaptation d'un outil open source peuvent s'avérer aussi chers que l'achat d'une licence d'un produit commercial qui répond à la totalité du besoin. Par contre il y a ce qu'on appelle la communauté open source, qui est le rassemblement de développeurs autour d'un projet, chacun chez soi, et qui participe souvent à une évolution significative d'un produit. Choisir un produit open source permet aussi de ne pas être dépendant d'un fournisseur qui peut changer de politique, ou se faire racheter, ce qui peut avoir une influence sur les contrats et sur la pérennité d'un produit.

Il existe une communauté française pour les produits open source, qu'on retrouve sur le site http://wiki.monitoring-fr.org/. La « Communauté francophone de supervision libre » s'intéresse particulièrement à Nagios qui est le plus connu des outils libres.

Les outils open source permettent d'être adaptés en fonction du besoin, afin d'adapter tel ou tel outil selon les besoins de l'organisation. Cela évite de faire les développements de base. Attention à ne pas croire que cela peut répondre à tous les besoins, car faire évoluer un produit open source peut coûter cher. À savoir, qu'il est nécessaire de compter le coût de développement, de maintenance, et la nécessiter d'assurer la compatibilité avec le produit open source qui continue à évoluer.

Voici une liste des solutions open source ou gratuites utilisées au CTI :

[28] Mesure 28 : mesure n°28 de réduction des dépenses http://ot.geneve.ch/ot/article.php3?id_article=73

[29] Plan de mesure du 29.11.2006 :
http://www.ge.ch/conseil_etat/2005-2009/communications/doc/ce061129-1.pdf

3.7.2.1 *Nagios et ses extensions*

Nagios est une solution puissante de supervision de milieu hétérogène permettant la surveillance système et réseau. Elle surveille les hôtes et services spécifiés, et lève des alertes lorsqu'elle détecte une anomalie sur un hôte, et informe également que la situation est revenue à la normale. Nagios est une base déjà complète, mais il existe de nombreuses extensions.

Les extensions, une centaine de minis programmes que l'on peut compléter en fonction des besoins de chacun pour superviser chaque service ou ressource disponible sur l'ensemble des ordinateurs ou éléments réseaux du SI. Le gros avantage de Nagios est qu'une solide communauté existe autour de cet outil et de nouvelles extensions sont régulièrement développées.

Les extensions utilisées sont notamment :

- Nagvis[30] : interface web de tableaux de bord
- Webinject[31] : permet de faire du test applicatif par requête HTTP/HTTPS entre autres.
- NConf[32] : interface web de configuration de Nagios (développé par Sunrise[33])

De base, tout s'édite à la main dans les fichiers de configuration de Nagios, il n'existe pas d'interface de gestion.

3.7.2.2 *Zabbix*[34]

Zabbix est plus récent que Nagios, et certains le préfèrent à ce dernier, car il est, semble-t-il, plus simple à appréhender. Zabbix peut travailler en mode distribué,

[30] Nagvis : interface web pour publier des tableaux de bord avec Nagios.

[31] Webinject : est une solution de test applicatif à l'aide de requête HTTP/HTTPS. Il existe une extension pour Nagios

[32] NConf : est une interface web de configuration de Nagios.

[33] Sunrise SA : est une société de télécommunication suisse.

[34] Zabbix : outil de surveillance open source.

mais ne permet pas d'agréger les données des serveurs distribués, ce qui en fait un outil limité pour les infrastructures importantes et complexes.

3.7.2.3 *Pingdom*[35]

Les services Pingdom sont proposés depuis un site web. Ça n'est donc pas un produit open source. La surveillance d'un seul site web est gratuite. Le concept est simple et ne requiert pas d'installation. Pingdom propose dans un premier temps un moteur d'analyse d'un site web afin d'en améliorer les performances, et dans un second temps une surveillance de la disponibilité du site, ou de certaines pages seulement. Un système d'envoi d'alerte par email ou par SMS (les 20 premiers sont gratuits) est également disponible.

Un produit tel que Nagios est bien moins onéreux à mettre en place qu'un produit équivalent développé en interne. Il est facile et rapide de mettre en place un outil open source et d'obtenir les premiers résultats. Par contre, cela va devenir vite plus compliqué dès que l'infrastructure s'étend, où que l'on souhaite l'utiliser comme solution transversale et ouverte à de nombreux utilisateurs. La solution actuellement au CTI ne gère que deux niveaux d'accès. Soit il y a un accès en lecture sur l'ensemble du produit, soit en écriture des fichiers de configuration sur le serveur.

Nagios, qui est utilisé pour plus de 99,5 % des tests du CTI, n'offre que la possibilité de faire toute la configuration par fichiers plats. La seule interface graphique qui existe de base ne permet que de consulter les alertes et les tableaux de bord. Des extensions sont parfois développées par des utilisateurs pour répondre à leurs besoins, mais même s'il est possible de télécharger ces extensions gratuitement, ils ne répondent pas nécessairement à nos propres besoins. Il serait donc nécessaire de les réadapter systématiquement. Nagios est bien adapté pour faire de la surveillance d'infrastructure, mais est trop contraignant à utiliser pour du test applicatif, bien qu'il existe aussi quelques extensions pour cela aussi, mais uniquement en requête HTTP, et sans simulation d'un utilisateur.

[35] Pingdom : site web offrant des services de surveillance.

Dans tous les cas, les outils open source ne permettent pas de faire des tests end-user, ou alors sont vraiment très limités.

3.7.3 Opter pour une solution commerciale qui couvre l'ensemble du besoin

Il existe des solutions commerciales qui couvrent la totalité du besoin, tant au niveau des tableaux de bord que des tests. La problématique de ce choix est de devenir dépendant d'une solution et de devoir l'imposer à tous les services. En règle générale, les solutions commerciales couvrent un périmètre plus large que les solutions open source. Les deux principaux avantages d'une solution commerciale sont les outils de test end-user, ainsi que les écrans de tableau de bord qui sont plus complets.

Ces solutions s'intègrent généralement avec une CMDB du marché. Elles sont toutes partagées en plusieurs couches : interface, moteur de traitement des données et persistance. La majorité s'intègre les unes avec les autres, même s'il est recommandé de choisir une solution complète. La plupart offrent un connecteur pour Nagios.

Quatre sociétés se partagent la majorité du marché. Elles sont surnommées les « big 4 », il s'agit de HP, IBM, BMC Software et CA Technologies.

Le seul argument contre le choix d'une solution commerciale réside sur le coût des licences. En effet, si chaque solution a ses spécificités dans le calcul du prix des licences, les principes de calculs restent communs. Cela se calcule au nombre de processeurs, sur le nombre de serveurs « robots » achetés, au nombre d'applications surveillées, au nombre d'alertes levées, ou encore sur un principe de point par type de test.

Dans tous les cas, plus on souhaite surveiller de serveurs et d'applications du système d'information, plus la solution coûte cher. Cela impose de faire des choix sur les éléments à surveiller, et de ne pas surveiller la totalité du SI.

3.7.4 Adopter une solution intermédiaire qui soit un mixte des trois

La raison qui nous pousse à envisager une solution mixte est un coût important pour une

solution qui soit uniquement basée sur un outil de surveillance commerciale. Ces dernières ont un avantage clé sur la partie tableaux de bord et sur les tests end-user. Par contre, une solution open source comme Nagios convient très bien pour la surveillance de composants systèmes. Les deux architectures s'intègrent l'une à l'autre, car les solutions commerciales proposent toujours une interface, payante, avec Nagios.

4 Réalisation technique d'un POC dans le contexte du CTI

Ce chapitre décrit la réalisation de la mise en œuvre d'un prototype qui démontre les concepts énoncés au précédent chapitre. Ce prototype a la contrainte de devoir tourner sur l'infrastructure existante du CTI.

4.1 Le Centre de Technologies de l'Information

Le Centre des Technologies de l'Information est le service informatique du canton de Genève, créé en 1997 qui emploie à ce jour environ 600 collaborateurs. Il est chargé de concevoir, choisir, développer, mettre en œuvre, exploiter, maintenir, renouveler de manière centralisée et gérer l'ensemble des moyens informatiques et de télécommunications dont il assure la pérennité et la sécurité.

À ce jour, il est composé de quatre Directions :

- le **Pôle Client** gère le portefeuille des projets et prestations.
- le **Centre de Solutions** a la responsabilité de concevoir, délivrer et maintenir des solutions exploitables pour les clients.
- le **Centre Infrastructures** garantit la pérennité du socle technologie (réseau, serveurs et applications techniques)
- le **Centre de Service** a la charge de satisfaire les utilisateurs (poste de travail et assistance utilisateur)

Le CTI est au service de sa maîtrise d'ouvrage, à savoir les sept départements politiques de l'État de Genève, ainsi que la Chancellerie d'État. http://www.ge.ch/organisation/

Le système d'information de l'État de Genève est actuellement constitué d'environ 2000 applications métiers tournant sur un peu plus de 2500 serveurs. Chaque application a un cycle de vie commun, à savoir une première étape de développement, de recette utilisateur, puis finalement de mise en production. Il s'agit de surveiller uniquement

l'environnement de production.

La multitude de serveurs se compose principalement de serveurs Windows 2000, Windows 2003, Windows 2010, de Linux CentOS, Linux Debian, Linux Redhat, d'HPUX, d'AIX, de Solaris 9 et Solaris 10. Chaque application peut être soit autonome, soit tourner derrière un reverse proxy. Elle peut nécessiter divers composants techniques comme une base de données, un annuaire, un service éditique, des webservices, des EJBs, des batchs ; elle peut être rattachée à d'autres applications ou encore être ouverte à l'extérieur du réseau de l'État de Genève.

À ce jour, la majeure partie des briques techniques sont surveillées, mais de manière complètement indépendante les unes des autres. Il s'agit principalement de serveurs ou d'éléments du réseau, il y a peu d'application, et encore moins avec l'ensemble de la chaîne dont elle dépend. De nombreux éléments sont surveillés par des services différents, qui ne partagent pas leurs informations.

En quelques chiffres[36], le CTI en 2010 c'est :

- 60'000 prises réseau
- 32'000 appels à la centrale d'assistance (internes et AEL[37] confondus) par an
- 10'000 demandes de support par an
- 400'000 spams détectés par jour
- 1000 serveurs physiques
- 1500 serveurs virtuels
- 2000 applications (métiers et techniques)
- 25'000 comptes utilisateurs (administration et secteur pédagogique)
- 10'000 comptes externes (citoyens et entreprises)
- 10'000'000 d'authentification sur le socle sécurité GINA[38]

[36] Ces chiffres sont tirés de http://www.ge.ch/cti/cti-chiffres.asp (2008), certains ont été mis à jour.

[37] AEL : Administration En Ligne. Services offerts aux citoyens et entreprises

[38] GINA : Gouvernance des Identités Numériques et des Accès. Socle sécurité de l'État de Genève

- Dont 200'000 authentifications depuis l'extérieur avec un mode d'authentification forte[39]

4.1.1 Les éléments surveillés au CTI

Ceci est un état de situation de la surveillance au CTI au printemps 2010.

4.1.1.1 *Réseaux et télécommunication (Direction infrastructures)*

Le service RT possède un serveur Nagios depuis près de 8 ans, qui teste par ping[40], SNMP[41] et à l'aide de scripts Perl[42] toute l'infrastructure réseau. A savoir l'ensemble des switchs[43], routeurs[44], access-points[45], proxies[46], reverse-proxies[47], serveurs Radius[48] et firewalls[49].

[39] Authentification forte : authentification avec un moyen qu'on possède en plus de couple utilisateur / mot de passe.

[40] Ping : commande informatique permettant de contrôler la communication entre deux machines d'un réseau.

[41] SNMP : Single Network Management Protocol et un protocole de communication qui permet de gérer des équipements réseau à distance.

[42] Perl : langage de programmation informatique interprété.

[43] Switch : élément simple reliant plusieurs segments d'un réseau.

[44] Routeur : élément complexe reliant plusieurs segments d'un réseau et permettant de faire de la redirection de paquets.

[45] Access-point : point d'accès WiFi, permettant à un PC de se connecter sans fil au réseau.

[46] Proxy : serveur servant de relai entre un client et un serveur. Il permet notamment de contrôler les flux sortants, par exemple comme passerelle de sortie sur internet pour une entreprise.

[47] Reverse-proxy : fonctionne comme un proxy, mais de manière inversée. Est utilisé par exemple pour publier un site web en entreprise, afin de définir un point d'accès central pour une ensemble d'utilisateur et en contrôlant les flux entrants.

[48] Serveur radius : offre un service d'authentification

[49] Firewall : un part-feu réseau est un élément du réseau permettant de définir des règles de sécurité d'accès entre deux réseaux

4.1.1.2 *Environnements documentaires et collaboratifs (Direction infrastructures)*

Le service EDC utilise la solution Microsoft SCOM[50] 2007 pour sa surveillance système, cela inclut notamment les serveurs Microsoft Exchange. Pour surveiller les passerelles SMTP sous Linux, ils utilisent les services de l'infrastructure Nagios mutualisée du Service S3[51] qui permet de contrôler :

- La gestion des queues SMTP[52]
- La version de l'antivirus
- Les temps de réponses SMTP
- La réception des alertes SNMP déclenchées par un contrôle des archivages d'e-mails

4.1.1.3 *Administration En Ligne*

Le service AEL surveille une partie de ses applications ainsi que son socle technologique à l'aide de l'infrastructure Nagios mutualisée du Service S3. L'AEL est à l'origine d'un projet au début 2010, nommé SI-Alert qui utilise Nagvis comme interface utilisateur. L'objectif du projet SI-Alert est d'avoir une vue synthétisée de la disponibilité de certaines applications stratégiques et critiques. Ces vues ont été ouvertes à quelques personnes du projet d'Administration en Ligne.

Le problème majeur de cet outil actuellement au CTI est qu'il n'est pas sécurisé. C'est-à-dire quiconque ayant accès à Nagios peut consulter l'ensemble des écrans Nagvis. Le pendant est également vrai, à savoir que chaque utilisateur ayant accès à Nagvis, peut également consulter l'ensemble des données techniques sous Nagios. Cet outil va pourtant évoluer, et cela peut être considéré comme une faille importante de laisser toutes ces données en accès aussi ouvert. Sécuriser ces

[50] Microsoft SCOM : solution de surveillance Microsoft.

[51] Service SSS : Serveurs, Systèmes et Stockage. Est un service du CTI, rattaché à la Direction Infrastructure, qui gère l'ensemble des serveurs.

[52] SMPT : protocole de communication permettant l'envoi d'email

écrans avec une gestion des rôles spécifiques sera un des résultats du prototype développé dans le cadre de ce mémoire.

4.1.1.4 *Gestion des Données et de l'Information*

Le service GDI, qui est responsables de toute l'infrastructure de base de données, constituée principalement des solutions Oracle[53], Microsoft SQL Serveur[54] et MySQL[55], utilise également l'infrastructure Nagios mutualisée. Plus de 180 serveurs et 350 bases de données sont ainsi surveillés.

4.1.1.5 *Serveurs, Systèmes et Stockage (S3)*

Le service S3 a mis en place un premier serveur Nagios il y a maintenant plus de 4 ans. Depuis le besoin ayant évolué, ils ont su monter une infrastructure Nagios complexe et mutualisée qui permet de surveiller d'autres services que le leur et surveille notamment :

- Applications web (HTTP et HTTPS) à l'aide de webinject et JMX
- 2500 serveurs physiques et virtuels
- 8000 services

4.1.1.6 *Solution BMC Patrol End-to-end Response Timer*

La solution PATROL[56] n'est plus maintenue par la société BMC, c'est pourquoi le CTI a entrepris le projet ARGOS afin de lui trouver un successeur.

Jusqu'à ce jour, cette solution était la plus utilisée comme surveillance applicative. Elle génère un point de situation quotidien sur un certain nombre de services et applications. Cette information est également disponible pour la MOA et régulièrement mise à jour sur le site Intranet de la Direction Infrastructures. PATROL permet de contrôler notamment des applications et réseaux de :

[53] Oracle : Entreprise spécialisée notamment dans les systèmes de base de données.

[54] MS SQL Serveur : Système de gestion de base de données Microsoft.

[55] MySQL : Système de gestion de base de données gratuite, du monde libre.

[56] PATROL : Solution de surveillance applicative de la société BMC. Ce produit est a été arrêté.

- La Police
- L'Administration fiscale
- L'Assurance maladie
- Le Service des automobiles et de la navigation
- L'Office cantonal de la population
- Le Système d'information des ressources humaines
- La Comptabilité financière intégrée

4.1.1.7 *Services Pingdom*

Certains utilisateurs ont utilisé les services de Pingdom sur des sites de l'État de Genève. Ces tests seront remplacés par le projet ARGOS.

4.1.1.8 *Nexthink Reflex v2*

La solution Reflex v2 de Nexthink[57] permet de faire une surveillance comportementale du poste de travail d'un utilisateur. Il va permettre de détecter qu'un service est lancé ou non sur un poste. Cette solution se démarque des autres solutions de surveillance, car l'objectif de Reflex est de détecter des applications ou services qui se lancent alors que ce n'est pas prévu. Il n'a pas pour objectif de contrôler la disponibilité d'un service.

Dans sa version 3, Nexthink a développé un module pour Nagios qui permet de :

- faire un suivi des SLA avec un système de notification par email
- connaître le nombre d'utilisateurs d'un service
- statut en temps réel
- génération de rapport automatique
- statistiques sur le taux d'utilisation des composants en relation avec les SLA

Mais ces informations ne répondent pas aux besoins du CTI.

[57] Nexthink : Solution de surveillance

4.1.1.9 *Surveillance des annuaires avec Zabbix*

Zabbix est l'outil utilisé pour surveiller les annuaires. À ce jour 25 annuaires Active Directory et 4 annuaires LDAP d'authentification sont surveillés. Zabbix permet de faire bien plus que de surveiller les annuaires, mais il n'est utilisé que par l'équipe d'ingénieurs responsable des annuaires.

4.1.1.10 *Surveillance des salles machines*

La surveillance des salles machines se fait avec l'outil Knürr RMS[58]. C'est un outil dédié à ce genre de surveillance, capable de mesure la température et l'hygrométrie d'un environnement informatique. Un système d'alerte par email ou par SMS permet de réagir rapidement. Cet outil peut également activer automatiquement des ventilateurs et autres matériels électriques de gestion des bâtiments.

4.1.1.11 *Surveillance des applications Java J2EE*[59]

Un développement a été fait en interne CTI pour permettre d'exécuter quelques tests techniques propres aux applications J2EE. C'est un composant de diagnostic, nommé Diag, qui peut être rattaché à toutes les applications Java J2EE développées par le CTI. Il permet de contrôler certains fichiers de paramétrage de l'application, ainsi que l'accès à la base de données ou encore à des webservices.

4.1.2 Le projet ARGOS

Le CTI a commencé en été 2008, il y a plus de 2 ans, à réfléchir sur une solution pour remplacer la solution BMC PATROL qui est actuellement la solution la plus largement utilisée pour la surveillance applicative, mais qui est devenue obsolète et n'est plus maintenue par le fournisseur.

---

[58] Knürr RMS : outil de surveillance des composants des salles machines, comme l'alimentation générale ou la température de la pièce. http://www.knuerr.com

[59] J2EE : Java Enterprise Edition est une spécification de Sun (www.sun.com) pour définir une approche structurelle multiniveaux pour les applications Java. Une application Java J2EE tourne sur un serveur applicatif dans une infrastructure distribuée.

Depuis 2008, l'objectif a évolué et l'ambition de ce projet est de trouver un système capable d'offrir une surveillance matérielle et applicative complète. De manière complémentaire, le nouvel outil devra être capable de récupérer l'ensemble des informations actuellement collectées afin de les consolider, les filtrer et les corréler. Ce projet, baptisé ARGOS en référence aux balises du même nom, a finalisé le cahier des charges fin 2009 et a lancé un appel d'offres public d'un budget de CHF 600'000.- en été 2010. Le dépouillement a lieu cet autonome pour arriver à une décision avant la fin de cette année.

4.1.3 Le projet SI-Alert

Au début 2010, un autre projet de surveillance à vu le jour en attendant les résultats du projet ARGOS. Ce second projet, nommé SI-Alert avait juste pour objectif d'avoir une vision de la disponibilité des services de l'Administration en Ligne à l'aide de quelques tableaux de bord. Si l'idée est excellente, la solution actuelle n'offre malheureusement aucune gestion des droits et donne toutes les informations à l'ensemble des utilisateurs ayant accès à Nagvis, même pour un autre besoin.

Le périmètre de ce mémoire est de démontrer la faisabilité d'une solution qui permet d'offrir à l'ensemble des utilisateurs, et surtout avec des droits restreints, des tableaux de bord de la disponibilité du système d'information. Pour démontrer cette faisabilité, un POC a été développé sur les bases du projet SI-Alert, pour surveiller l'application de gestion des droits, et l'ensemble des composants dont elle dépend.

4.2 POC : Méta-surveillance de l'application « GINA.Manager ».

4.2.1 But du POC

Ce POC doit démontrer la faisabilité, ou non, du modèle selon les contraintes du CTI. L'objectif d'un POC n'est pas de tout réaliser, mais de poser les bases d'un modèle de méta-surveillance, avec un tableau de bord utilisateur et les tableaux de bord pour les

concepteurs.

L'objectif est de fournir une surveillance, à l'aide de composants open source, de l'application de gestion de la sécurité utilisée à l'État : Gina.Manager. Pour que cette application J2EE fonctionne, de nombreux composants doivent également fonctionner. Il faut donc définir une vue utilisateur qui ne verra qu'une seule application, et les tableaux de bord des concepteurs composants qui permettrons de consulter le détail de l'ensemble des dépendances de l'application surveillée.

4.2.2 Les différentes étapes de la réalisation

1. Choix des outils de surveillance
2. Architecture de l'application « Gina.Manager »
3. Tests applicatifs et tests de composants avec Nagios
4. Intégration des rôles de bases dans Nagvis
5. Tableau de bord utilisateurs avec l'extension Nagvis
6. Tableaux de bord concepteurs avec l'extension Nagios Business Process
7. Test de validation du POC

4.2.3 Choix des outils de surveillance

4.2.3.1 *Moteur de test*

Actuellement tout nouveau composant à surveiller est intégré à l'infrastructure mutualisée de Nagios mise en place depuis près de 8 ans au CTI. Au début, il ne s'agissait que de surveillance localisée, mais depuis 2 ans cela devient l'outil central de surveillance. En attendant les résultats du projet ARGOS, dont l'installation débutera au début 2011. C'est donc Nagios, utilisé à plus de 99 % au CTI, qui nous servira de moteur de test.

L'architecture de Nagios actuellement au CTI est composée d'un serveur central, le Master. Il consolide les informations reçues de deux autres Nagios, un pour chaque zone réseau, qui servent d'ordonnanceurs et sur lesquels sont stockés

toutes les règles et scripts de test.

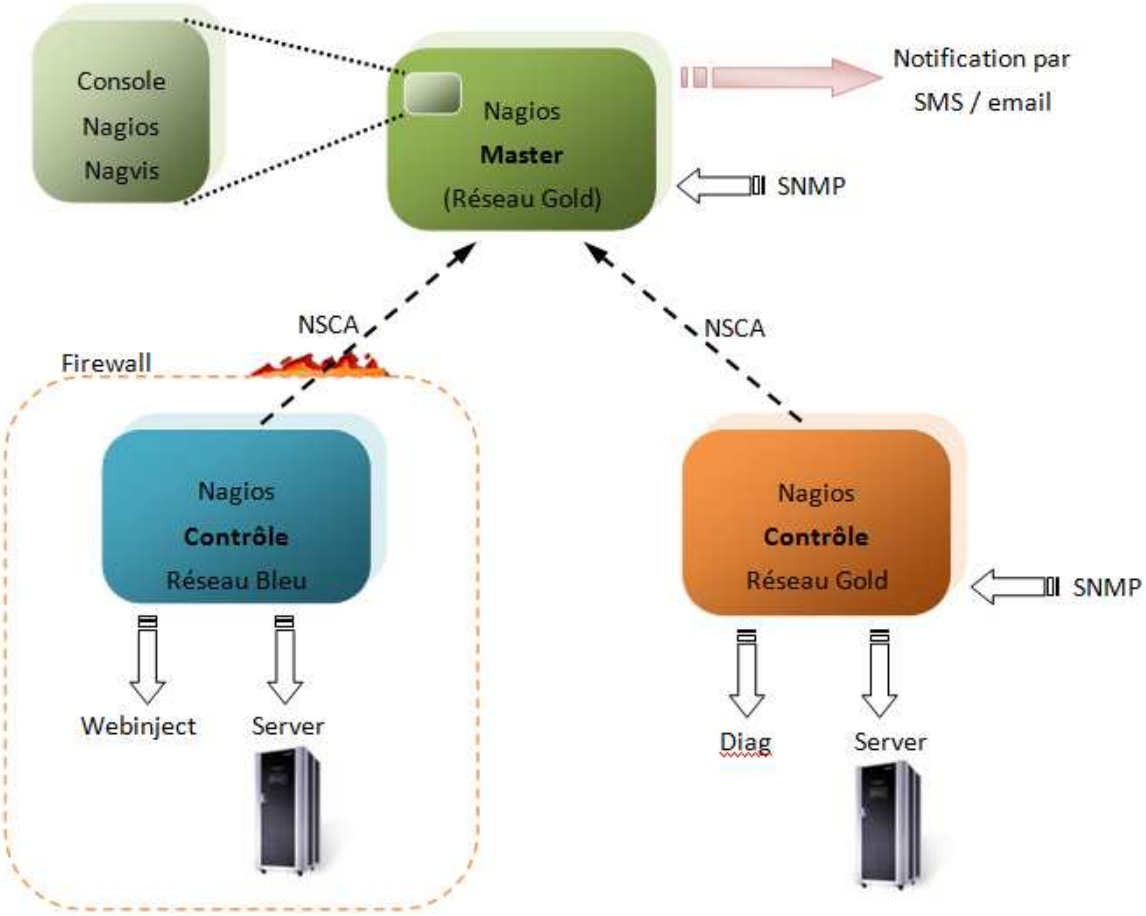

Figure n°14 - Architecture des serveurs Nagios pour scanner dans deux sous-réseaux distincts

- **NSCA** : Nagios Service Check Acceptor. Service permettant de faire du test passif sur des machines distantes.
- **Webinject** : Extension Nagios permettant de faire des tests applicatifs par requête HTTP.

- **Server** : Tests du système avec un agent NRPE[60] installé sur les serveurs qui permet d'exécuter des tests à la demande du serveur Nagios. Les scripts sont stockés en local sur le serveur.
- **Diag** : Composant Java développé au CTI permettant de faire du contrôle d'application Java.
- **SNMP** : Protocole utilisé quand il n'est pas possible d'installer un agent sur l'élément à contrôler.

Le serveur Nagios utilisé pour faire le POC est un serveur de laboratoire sur lequel sont testés les scripts avant que ceux-ci soient installés sur l'architecture mutualisée.

4.2.3.2 *Interface utilisateur*

Comme vu dans la sous-section *3.7.2.1* ci-dessus - *Nagios et ses* , il existe de nombreuses extensions d'interfaces pour Nagios. Le projet SI-Alert utilise depuis 6 mois une interface avec Nagvis qui permet d'avoir une vue web paramétrable pour les tableaux de bord utilisateurs. Nagvis est une extension open source de Nagios qui est la plus aboutie du marché actuellement. Il existe même de nombreuses extensions qui s'appuient sur Nagvis pour en faire des évolutions.

Dans le cadre de ce POC, Navgis est utilisé sans extension complémentaire. Comme il s'agit d'un produit open source, nous pouvons y faire les évolutions souhaitées sans contrainte. Le but est d'intégrer ces évolutions de manière industrialisable, c'est-à-dire qu'en cas de mise à jour du produit, il soit facile et surtout possible de reporter ces évolutions. La seule manière propre de le faire est en surchargeant les classes et les interfaces existantes.

Nagvis annonçait[61] le 7 mars 2010 sa première version bêta[62] 1.5b1 intégrant une refonte des mécanismes d'authentification et d'autorisation. Nous avons dû

[60] NRPE : Nagios Remote Plugin Executor. Agent Nagios installé sur le poste surveillé, afin d'exécuter des tests locaux, comme la surveillance de la température du processus ou de la charge mémoire.

[61] Annonce faite sur son blog http://www.nagvis.org/home avec le titre « NagVis 1.5b1 released »

attendre le 15 juin pour qu'elle soit définitivement disponible en version stable 1.5.0. La dernière version utilisée pour le POC est la version 1.5.3

4.2.3.3 *Système d'alerte*

Le système d'alerte retenu est soit un email envoyé à un point de contact, soit un SMS. L'envoi de SMS se fait avec une solution propriétaire du CTI.

4.2.3.4 *Stockage des données*

La persistance des données est faite sous forme de fichier plat de configuration, ainsi qu'avec SQLite, un système de gestion de base de données embarqué dans Nagios.

4.2.4 Architecture de « Gina.Manager »

Dans le cadre de ce POC, il s'agit de surveiller l'application de gestion des accès. Les composants surveillés sont ceux de l'environnement de test, c'est pourquoi tous les éléments ne sont pas en haute disponibilité.

Pour qu'un utilisateur puisse accéder à l'application Gina.Manager, il doit préalablement être authentifié. L'authentification passe par plusieurs étapes : un reverse proxy, un radius, puis finalement un annuaire en cluster.

Après s'être authentifié, si l'utilisateur possède les autorisations nécessaires, le reverse proxy le redirige vers l'application, qui tourne sur un serveur applicatif. À chaque accès spécifique, un contrôle est effectué sur le provider de sécurité.

Pour fonctionner, une application J2EE telle que celle-ci a donc besoin de cinq composants distincts, dont un installé en cluster. Des tests applicatifs peuvent également être produits afin de valider également le fonctionnement end-user.

[62] Version bêta : est une pré-version que les utilisateurs peuvent tester afin d'y rapporter des anomalies. Une version bêta n'est pas stable et ne doit pas être utilisée dans un environnement de production.

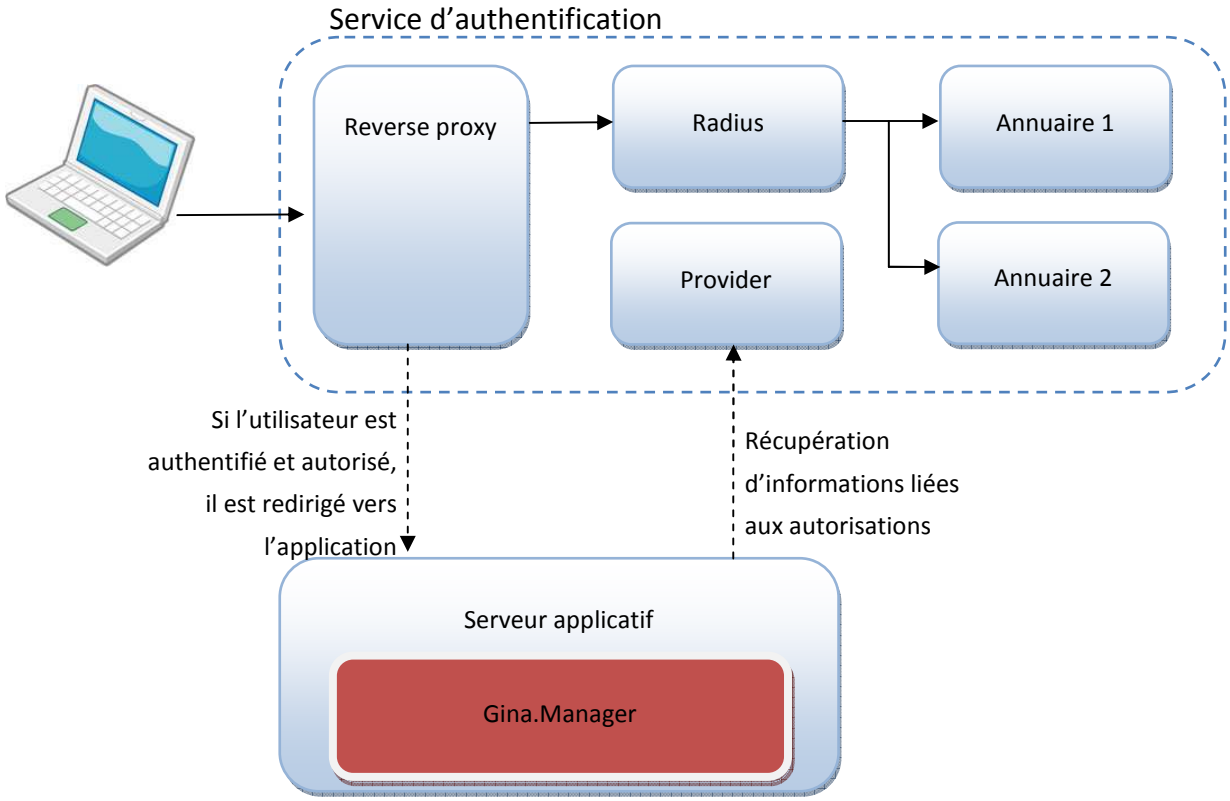

Figure n°15 - Vue conceptuelle des composants de Gina.Manager utilisés pour le POC

4.2.5 Tests applicatifs et tests des composants avec Nagios

Dans Nagios, les fichiers de configuration se trouvent dans le répertoire `/nagios/etc/conf.d` et se terminent par l'extension `.cfg`. Tous les tests du POC, se configurent dans un seul fichier nommé `vot.cfg`. Dans notre cas, nous testons simplement l'état du serveur, ainsi que le service générique `ping`.

4.2.5.1 *Test du reverse proxy*

```
define host{
        use                     generic-host
        host_name               devtech.etat-ge.ch
        alias                   devtech.etat-ge.ch
        address                 10.138.40.88
        check_command           check-host-alive-tcp-443
```

```
}
define service{
     host_name              devtech.etat-ge.ch
     service_description    ping
     check_command          check_ping
     use                    generic-service
}
```

4.2.5.2 *Test du serveur radius*

```
define host{
     use             generic-host
     host_name       radius1
     alias           romer2
     address         10.138.41.29
}
define service{
     host_name              radius1
     service_description    ping
     check_command          check_ping
     use                    generic-service
}
```

4.2.5.3 *Test des annuaires*

```
define host{
     use             generic-host
     host_name       annuaire1
     alias           corbeau
     address         10.137.208.119
}
define host{
     use             generic-host
     host_name       annuaire2
     alias           scheat
     address         10.137.208.154
}
define service{
     host_name              annuaire1,annuaire2
     service_description    ping
     check_command          check_ping
```

```
    use                     generic-service
}
```

4.2.5.4 *Test du provider*

```
define host{
    use             generic-host
    host_name       provider1
    alias           bray2
    address         10.138.41.37
}
define service{
    host_name               provider1
    service_description     ping
    check_command           check_ping
    use                     generic-service
}
```

4.2.5.5 *Test applicatif*

```
define service{
    host_name               devtech.etat-ge.ch
    service_description     ginamanager
    check_command           webinject!vot_conf.xml
    use                     generic-service
}
```

Ceci est complété par les informations du fichier `vot_conf.xml` du répertoire `/nagios/etc/conf.d/webinject/`.

```
<reporttype>nagios</reporttype>
<testcasefile>vot.xml</testcasefile>
<timeout>10</timeout>
<globaltimeout>20</globaltimeout>
```

Qui lui-même référence le fichier `vot.xml` du même répertoire. Ce fichier exécute deux tests, le premier pour récupérer la session et le deuxième pour l'authentification.

```
<testcases repeat="1">
```

```
<case
    id="08888 »
    description1="Page de redirection vers GINA sur l'apache
interne »
    description2="Presence JSESSIONID »
    method="get »
    URL="https://prod.etat-ge.ch/ginalogin/int-admin-
prod/start?path=dummy »
        verifypositive1="JSESSIONID"
        parseresponse='st » value="|"|escape'
/>

<case
    id="660601 »
    description1="Authentification GINA »
    description2="Variable st={PARSEDRESULT} »
    method="post »
    URL="https://prod.etat-ge.ch/ginalogin/int-admin-prod/login »
        postbody="st={PARSEDRESULT}&username=COMPTE-
TEST&password=MOT-DE-PASSE »
        addheader="Accept :
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8 »
    verifypositive1="ETATRAD »
    verifynegative1="error »
/>

</testcases>
```

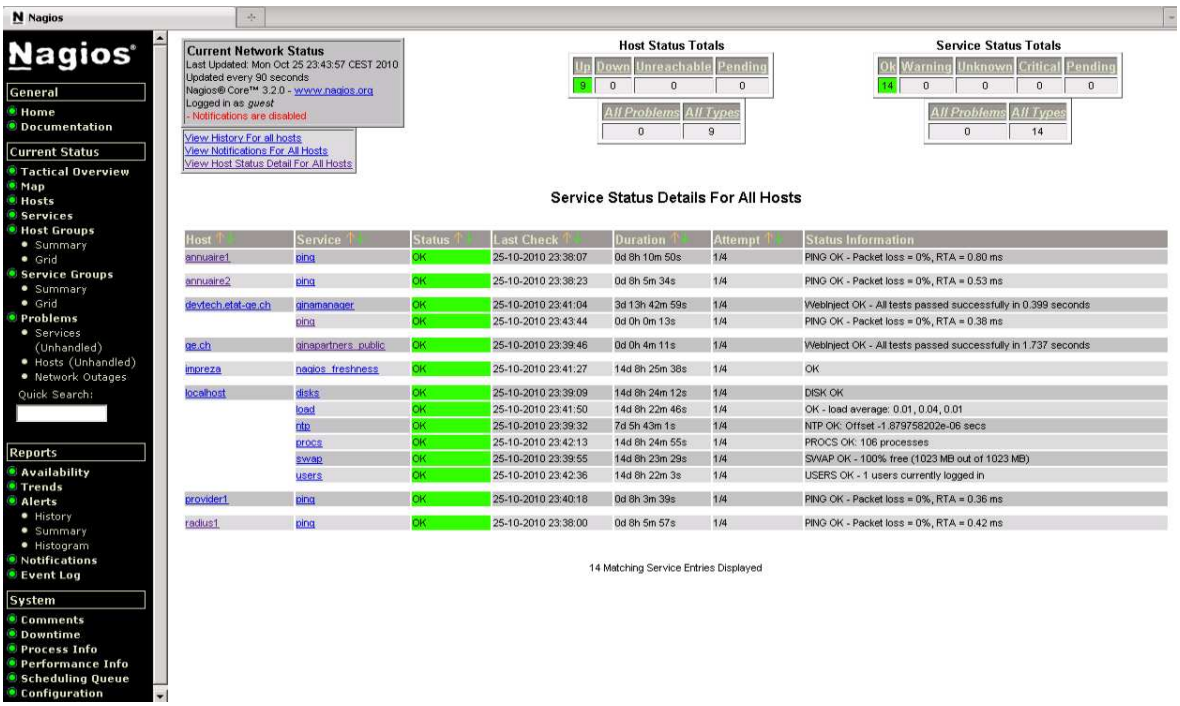

Figure n°16 - Vue globale de Nagios, menu Services

Figure n°17 - Zoom de la vue globale sur les composants du POC

Ce que l'on peut constater, c'est que Nagios considère tous les différents composants à plat. La seule chose que l'on puisse voir, c'est lorsque plusieurs tests sont faits pour un composant, sinon aucune notion de dépendance n'existe. Cette vue n'est évidemment pas présentable telle quelle pour un utilisateur.

4.2.6 Intégration des rôles de base dans Nagvis

Selon le modèle, sept rôles de base ont été définis. Dans le cadre de ce POC, quatre rôles ont été implémentés dans Nagvis afin d'avoir des restrictions en fonction des droits définis dans le référentiel de gestion des accès Gina.Manager.

- **Utilisateur** : ne voit que la disponibilité de l'application, sans possibilité de consulter les composants dont elle dépend.
- **Concepteur** : accède à l'ensemble des écrans et peut également modifier les tableaux de bord
- **Surveillance** : accède à l'ensemble des écrans en lecture uniquement.
- **Administrateur** : accède en écriture à la totalité de Nagvis. Peut créer ou supprimer des tableaux de bord

Les autres rôles développés dans le modèle de Meta-surveillance ne sont que des évolutions ou des particularités de ces quatre rôles de base.

Dans Nagvis, il a fallu modifier le fichier *nagvis.ini.php* qui se trouve dans le répertoire */nagvis/etc/* , afin d'activer le déport d'authentification et la gestion des autorisations sur des classes surchargées. Les attributs suivants ont été modifiés :

```
authmodule="CoreAuthModGina »
authorisationmodule="CoreAuthorisationModGina »
logonmodule="LogonGina"
logonenvvar="REMOTE_USER »
```

Ensuite, les fichiers du répertoire */nagvis/share/server/core/classes/* qui ont été surchargés sont les suivants :

CoreAuthModGina.php
CoreAuthorisationModGina.php
CoreModLogonGina.php

Ainsi que dans le répertoire */nagvis/share/frontend/nagvis-js/classes/* :

FrontendModLogonGina.php

Puisque Nagvis s'appuie sur une base de données SQLite, il a fallu comprendre le modèle de données et modifier certaines requêtes comme par exemple le code suivant :

```
$RES = $this->DB->query('SELECT perms.mod AS mod, perms.act AS act,
perms.obj AS obj '.
'FROM users2roles '.
'INNER JOIN roles2perms ON roles2perms.roleId = users2roles.roleId
'.
'INNER JOIN perms ON perms.permId = roles2perms.permId '.
'WHERE users2roles.userId = '.$this->DB->escape($userId));
$RES = $this->DB->query($select);
```

Devient :

```
$GINAWS = new Gina();
$sRoles = $GINAWS->getRoles('CTI.SURVEILLANCE');
$select = 'SELECT perms.mod AS mod, perms.act AS act, perms.obj AS
obj '.
'FROM roles2perms '.
'INNER JOIN roles ON roles2perms.roleId = roles.roleId '.
'INNER JOIN perms ON perms.permId = roles2perms.permId '.
'WHERE roles.name = ';

$premier = true;
foreach ($sRoles as &$role) {
     if($premier){
                              $select .= '\''.$role.'\'';
                              $premier = false;
     }else{
                              $select .= ' OR roles.name =
\''.$role.'\'';
```

```
    }
}
$RES = $this->DB->query($select);
```

Cette intégration permet de gérer les comptes utilisateurs et les droits d'accès dans le référentiel sécurité de l'État de Genève. Il n'y a plus besoin de gérer les droits dans Nagvis.

Pour gérer les accès des utilisateurs, de manière globale et automatique, il est possible de s'appuyer sur le référentiel de gestion des accès en définissant un héritage entre les applications surveillées et le rôle utilisateur de la surveillance.

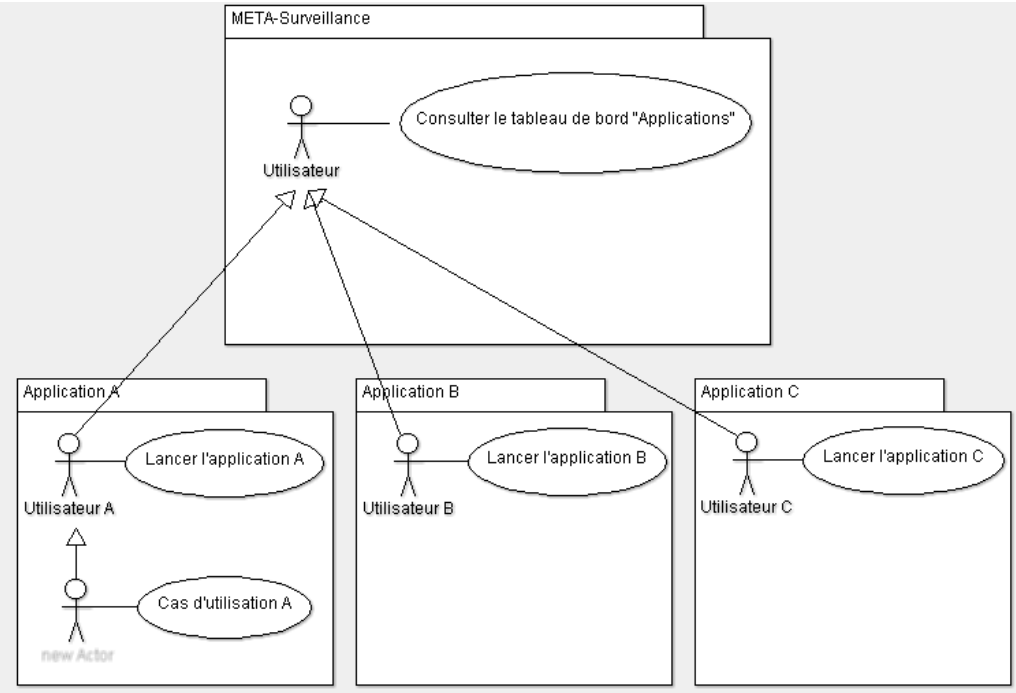

Figure n°18 - Diagramme de classe UML avec héritages du rôle Utilisateur entre différentes applications

De cette manière, en ajoutant un attribut « nom de l'application » dans le tableau de bord d'une application dans Nagvis, on affiche pour l'utilisateur connecté que les écrans pour lesquels il est utilisateur de l'application. À partir de ce moment, la totalité de la

gestion des droits se fait dans Gina, et l'intégration de Nagvis est terminée.

4.2.7 Tableaux de bord Utilisateur avec l'extension Nagvis

Nagvis est fait pour produire des tableaux de bord complets d'une application, avec l'ensemble des composants dont elle dépend sur une seule vue. Une utilisation standard de Nagvis nous amènerait à utiliser le schéma d'architecture (*Figure n°15 - Vue conceptuelle des composants de Gina.Manager utilisés pour le POC*) et d'y ajouter les composants surveillés. De cette manière Nagvis répond à l'utilisation de base qui correspondrait au rôle Concepteur.

Chaque indicateur vert correspond à un état de disponibilité. Les indicateurs de forme carrée correspondent à des contrôles du matériel, comme la disponibilité du serveur. Les indicateurs de forme ronde correspondent à la disponibilité d'un service applicatif tournant sur un serveur.

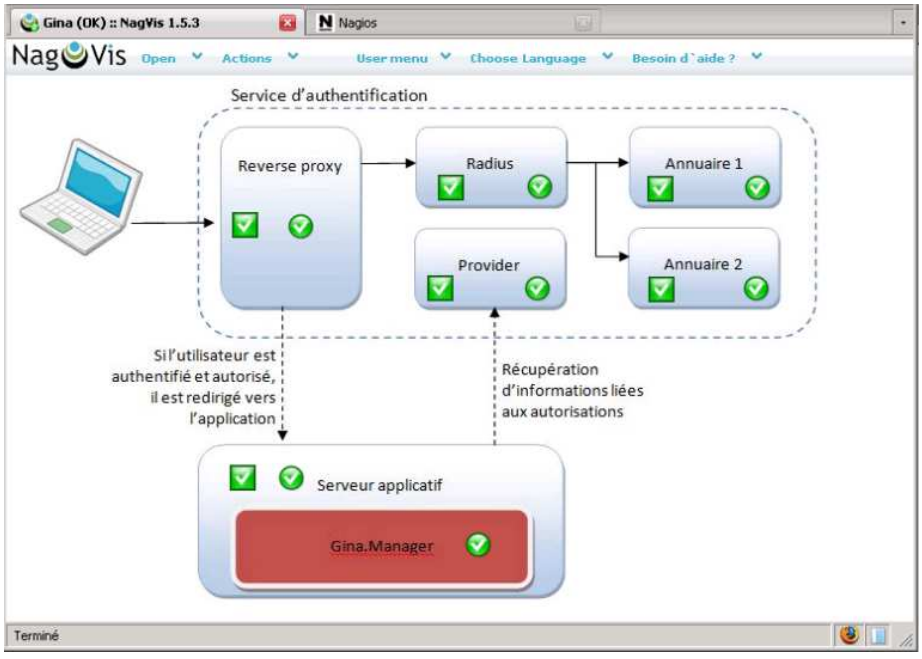

Figure n°19 - Nagvis - Vue concepteur

Pour le rôle Utilisateur, une vue complémentaire, avec un test de disponibilité de

l'application uniquement a été créée.

Figure n°20 - Nagvis – Vue utilisateur

Pour valider ce tableau de bord, une panne a été simulée en débranchant un composant, comme le serveur Radius par exemple. Voici le résultat du tableau de bord Concepteur :

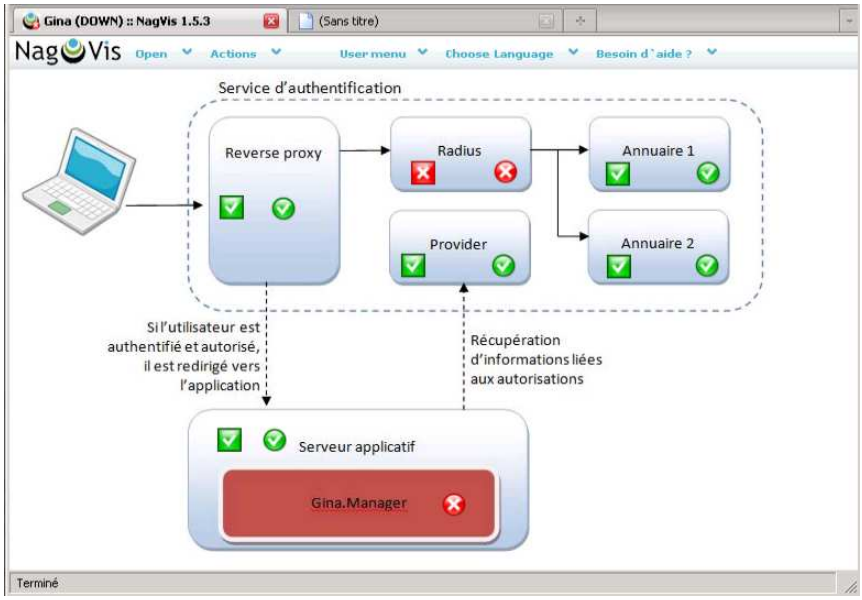

Figure n°21 - Nagvis – Vue Concepteur avec panne sur Radius

On constate que le serveur Radius n'est plus présent, et que le contrôle du serveur ne répond pas non plus. Le test de Gina.Manager qui est un test d'authentification ne peut

plus fonctionner, c'est pourquoi le test applicatif Gina.Manager échoue. La vue Utilisateur quant à elle s'appuie uniquement sur le service applicatif Gina.Manager, donc se trouve au rouge également.

Malgré le fait de pouvoir définir une vue Utilisateur indépendante d'une vue Concepteur, il est forcé de constater que Nagvis ne permet pas de définir une relation de dépendance entre les composants sur une seule vue. Puisque le reverse proxy ne change pas de statut. La seule manière qui permet de définir une relation de dépendance avec Nagvis consiste à faire une sous-vue pour chaque composant. C'est-à-dire de mettre le radius dans une vue qui se trouve dessous le reverse proxy. Les deux annuaires seraient quant à eux dans une troisième vue, sous le radius.

Pour simuler cet exemple, nous allons simplement tester le résultat en mettant la vue Concepteur comme dépendance sous la vue Utilisateur. La vue Concepteur ne change pas, et sur la vue Utilisateur a été ajouté un symbole carré représentant la dépendance sur la vue Concepteur placée au-dessous.

Figure n°22 - Nagios – Vue utilisateur avec dépendance, sans panne

Figure n°23 - Nagios – Vue utilisateur avec panne sur dépendance

Sur la première vue, quand tout fonctionne bien, on voit que l'indicateur rond du test applicatif ainsi que l'indicateur carré de la vue globale du dessous sont tous les deux au vert.

Sur la seconde vue, alors qu'un des clusters des annuaires a été coupé cette fois, l'indicateur

carré de la vue globale passe au rouge, alors que cela ne devrait rien changer. Il n'est pas possible avec Nagvis de gérer une notion de warning si un cluster tombe, puisque l'indicateur de dépendance passe au rouge si au moins un indicateur de la vue du dessous passe au rouge. Dans notre vue Concepteur, il y a treize indicateurs.

Cela permet néanmoins d'avoir une vision globale et de pouvoir constater immédiatement si une anomalie survient. Malgré tout, cette manière de faire comporte plusieurs désavantages. Le premier est la lourdeur de la mise en place, avec autant de vues à définir qu'il y a de composants et d'application, sachant que chaque application est à définir deux fois. Une fois pour l'Utilisateur avec pour unique indicateur l'indicateur applicatif. Et la seconde vue pour les Concepteurs avec un indicateur du service applicatif, complété de l'indicateur de dépendance sur la vue globale du dessous.

Le deuxième inconvénient est qu'avec cette structure de dépendance sur les vues, si un annuaire en cluster tombe, l'indicateur d'indisponibilité va remonter de vue en vue. Donc sur la vue la plus haute, il y aura un indicateur qui précise que l'application fonctionne et un autre qui précise que cela ne fonctionne pas. Nagvis ne permet pas de définir la notion de cluster, en remontant un état d'alerte de type warning, sans que cela soit un état d'indisponibilité total.

4.2.8 Tableaux de bord Concepteur, avec l'extension Nagios Business Process.

Nagvis n'offre aucune possibilité de faire des tableaux de bord concepteur avec la liste des dépendances. Pour cela, il existe une autre extension qui permet d'avoir cette liste de dépendance et qui offre même la possibilité de définir des cas de dépendance en cluster. Il s'agit de Nagios Business Process (NBP).

4.2.8.1 *Définition des dépendances*

Dans NBP, la définition des dépendances se fait dans le fichier `/nagiosbp/etc/nagios-bp.conf` à l'aide d'opérateur ET (&) et OU (|).

Deux services ont été définis :

1. Gina.Authentification, avec l'ensemble des composants nécessaires au bon fonctionnement.
2. Gina.Manager, qui englobe le test applicatif, ainsi que la dépendance sur Gina.Authentification.

```
annuaires = annuaire1;ping | annuaire2;ping
display 0;annuaires;Les Annuaires

radius = radius1;ping
display 0;radius;Services Radius

provider = provider1;ping
display 0;provider;Provider

reverseproxy = devtech.etat-ge.ch;ping
display 0;reverseproxy;Reverse Proxy

ginaauth = reverseproxy & radius & annuaires & provider
display 1;ginaauth;Gina Authentification

serveurapplicatif = serveur_applicatif;ping &
serveur_applicatif;ginamanager
dispay 0;serveurapplicatif;Serveur applicatif

ginamanager = ginaauth & serveurapplicatif
display 1;ginamanager; Gina.Manager
```

4.2.8.2 *Présentation des vues de NBP*

Cela donne la vue globale suivante :

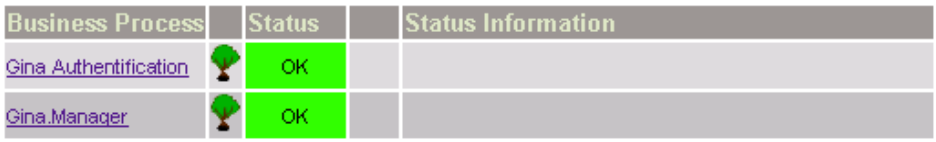

Figure n°24 - NBP – Vue globale, sans panne

En partant de cette vue, on peut afficher la liste de l'ensemble des composants en cliquant sur le lien de la colonne Business Process.

Host	Service	Status	Status Information
annuaire1	ping	OK	PING OK - Packet loss = 0%, RTA = 0.69 ms
annuaire2	ping	OK	PING OK - Packet loss = 0%, RTA = 0.45 ms
devtech.etat-ge.ch	ping	OK	PING OK - Packet loss = 0%, RTA = 3.40 ms
provider1	ping	OK	PING OK - Packet loss = 0%, RTA = 0.34 ms
radius1	ping	OK	PING OK - Packet loss = 0%, RTA = 0.37 ms

Figure n°25 - NBP – Liste des composants de Gina Authentification, sans panne

Ou alors afficher la liste des dépendances en cliquant sur le symbole ressemblant à un arbre. Le mot « and » signifie que pour que le service Gina Authentification fonctionne, il faut que l'ensemble des composants listés fonctionnent.

	Host	Service	Status	Status Information
and*	Reverse Proxy		OK	
	Services Radius		OK	
	Les Annuaires		OK	
	Provider		OK	

Figure n°26 - NBP – Liste des dépendances de Gina Authentification, sans panne

En cliquant sur « Les annuaires », on peut descendre encore plus bas dans la liste des composants et accéder à la liste des annuaires en cluster. Comme ils sont redondants, c'est l'opérateur « or » qui s'affiche.

	Host	Service	Status	Status Information
or*	annuaire1	ping	OK	PING OK - Packet loss = 0%, RTA = 0.69 ms
	annuaire2	ping	OK	PING OK - Packet loss = 0%, RTA = 0.45 ms

Figure n°27 - NBP – Liste des dépendances des Annuaires, sans panne

L'ensemble des écrans qui nous intéressent a été présenté. Voyons maintenant comme réagit l'extension Nagios Business Process lorsqu'un composant unique tombe en panne. Puis nous verrons comment réagit NBP lorsqu'un composant en cluster

tombe.

4.2.8.3 *Test en coupant un composant qui n'est pas en cluster*

En coupant un serveur radius, sans redondance, cela donne ça :

Business Process		Status		Status Information
Gina Authentification		CRITICAL		
Gina.Manager		CRITICAL		

Figure n°28 - NBP – Vue globale, avec panne sur Radius

	Host	Service	Status	Status Information
and*		Reverse Proxy	OK	
		Services Radius	CRITICAL	
		Les Annuaires	OK	
		Provider	OK	

Figure n°29 - NBP – Liste des dépendances de Gina Authentification, avec panne sur Radius

4.2.8.4 *Test en coupant un composant en cluster*

En coupant un annuaire en cluster, NBP montre qu'il n'y a donc pas d'impact au niveau de la vue globale, puisque l'élément qui a été coupé est un composant en cluster, et que le deuxième cluster fonctionne correctement.

Business Process		Status		Status Information
Gina Authentification		OK		
Gina.Manager		OK		

Figure n°30 - NBP – Vue globale, avec panne sur un annuaire

Sur la liste de l'ensemble des composants, on constate qu'un des annuaires en cluster est bien en état CRITICAL. Cette vue permet d'avoir une vision rapide et globale de l'ensemble des composants pour savoir immédiatement si un composant tombe en

panne.

Host	Service	Status	Status Information
annuaire1	ping	OK	PING OK - Packet loss = 0%, RTA = 3.97 ms
annuaire2	ping	CRITICAL	PING CRITICAL - Packet loss = 100%
devtech.etat-ge.ch	ping	OK	PING OK - Packet loss = 0%, RTA = 9.10 ms
provider1	ping	OK	PING OK - Packet loss = 0%, RTA = 0.34 ms
radius1	ping	OK	PING OK - Packet loss = 0%, RTA = 0.37 ms

Figure n°31 - NBP – Liste des composants, avec panne sur un annuaire

Quant à la vue des dépendances de Gina Authentification, tous les services fonctionnent correctement.

	Host	Service	Status	Status Information
and*	Reverse Proxy		OK	
	Services Radius		OK	
	Les Annuaires		OK	
	Provider		OK	

Figure n°32 - NBP – Liste des dépendances de Gina Authentification, avec panne sur un annuaire

Par contre, en consultant le détail des composants constituants le service des Annuaires, on constate que l'un des annuaires est bien en état CRITICAL.

	Host	Service	Status	Status Information
or*	annuaire1	ping	OK	PING OK - Packet loss = 0%, RTA = 3.97 ms
	annuaire2	ping	CRITICAL	PING CRITICAL - Packet loss = 100%

Figure n°33 - NBP – Liste des dépendances des annuaires, avec panne sur un annuaire

4.2.9 Discussion sur les résultats du POC

Ce POC démontre la faisabilité du modèle de méta-surveillance. Ce modèle est applicable sur des outils open source. Cependant il faut admettre que l'utilisation d'outils open source pour surveiller une infrastructure importante devient rapidement une tâche laborieuse.

L'utilisation des extensions open source de Nagios ne se fait pas sans rencontrer de difficulté, car il faut envisager d'utiliser une extension différente pour chacun de nos besoins. Certes, il est possible de sécuriser de manières indépendantes les accès aux extensions. De ce fait, l'extension Nagios Business Process ne pourrait être ouverte qu'aux Concepteurs, car cette extension n'intègre aucune notion de sécurité. Mais il n'est pas possible de corréler les extensions entres-elles et les fichiers de configuration doivent être réécrits à chaque fois. Certes, cela pourrait être fait à l'aide scripts, mais encore un fois, cela demande un investissement complémentaire.

Il serait tout à fait envisageable de développer une couche d'authentification et d'autorisation sur une extension. Cela représente malgré tout un risque de dépendance à la version utilisée pour intégrer cette évolution. Si ce n'est pas une évolution reconnue officiellement par le groupe proposant cette extension, il sera probablement nécessaire de réadapter la couche sécurité à chaque nouvelle version. Ceci est bien sûr un frein à l'utilisation d'outils open source que l'on voudrait faire évoluer.

Conclusion

À ce jour, la surveillance des systèmes d'information est arrivée à une certaine maturité. Il est possible de mettre en place une surveillance complète de son système informatique, ainsi que du système d'information et d'ouvrir ces données aux utilisateurs. La technologie n'est plus un frein à cette démarche. Le principal blocage de la diffusion des informations sur la disponibilité des SI reste la difficulté de communication et la crainte de ne pas pouvoir contrôler la manière dont elles pourraient être utilisées. Donner accès à ces informations est encore vu comme une forme de révolution, puisque ce n'est pas une pratique courante dans les entreprises.

Pourtant, c'est possible. Certes, cela nécessite un investissement initial considérable. Mais pour un retour sur investissement certain et qui va plus loin que le gain financier. Grâce aux tableaux de bord de la disponibilité de son système d'information, c'est avoir une mesure précise qui permet de valider les choix d'investissement sur l'infrastructure, et c'est notamment une manière de gagner la confiance de ses utilisateurs. C'est pourquoi il faut pouvoir s'appuyer sur des briques solides comme une CMDB ou un référentiel de gestion des accès.

Gagner la confiance des utilisateurs est possible en montrant de la transparence sur la disponibilité du système d'information. Cela nécessite un accompagnement sur le long terme et une communication sans faille. Car si un utilisateur constate une différence entre les tableaux de bord et la réalité, cela pourrait avoir comme conséquence de casser une relation entre l'utilisateur et le centre informatique déjà souvent fragile. Il est nécessaire d'expliquer aux utilisateurs que la surveillance est principalement un outil d'aide pour augmenter la qualité des services informatiques.

Du côté du centre informatique, se fixer comme objectif de partager les informations de disponibilité avec les utilisateurs nécessite comme première étape de partager ces informations entres les services du centre informatique. La réalisation de cette première étape est un déjà un facteur de gain de qualité assuré.

Les solutions open source permettent à toutes entreprises de commencer à mettre en place

une surveillance de son système d'information à moindres frais. Pour les organisations de grande taille, il est cependant nécessaire d'investir dans des solutions propriétaires qui sont les seules à offrir des outils de test applicatifs complets et des facilités de déploiement à large échelle. Des outils solides de modélisation de tests simulant le comportement d'un utilisateur final n'existent pas en open source. La mise en place de tableaux de bord corrélés entre eux et évolutifs est une des forces majeures des outils propriétaires. Il est évident que les coûts de licence sont importants, mais une étude approfondie des besoins de surveillance et de définition des priorités permet de conserver une partie de la surveillance avec des outils open source et de compléter le manque par les avantages d'une solution commerciale.

Ce mémoire m'a permis de comprendre les enjeux d'une surveillance globale d'un système d'information. Et surtout la complexité de l'ouverture aux utilisateurs. De plus, le sujet de l'approche de surveillance d'un SI dans sa globalité est toujours abordé en relation avec un outil. Très peu de documentation existe en ayant une approche indépendante et s'appuyant sur les référentiels existants dans l'entreprise. Avec ce mémoire, je souhaite pouvoir apporter une documentation complémentaire dans ce domaine.

Annexe A – Classification des environnements

La société Harvard Research Group (HRG) a défini la disponibilité comme étant l'impact d'une panne sur l'activité de l'entreprise et sur l'utilisateur final, selon une classification qui s'établit sur six niveaux. Lorsque survient un incident de fonctionnement et que les processus de récupération prévus l'ont pris en charge, le système d'information est supposé ne plus se trouver dans son état d'environnement disponible (AE = Availability Environnement), jusqu'à ce que la panne ait été réparée.

Tableau II. Classification des environnements

AEC-0	Standard	Les fonctions peuvent être interrompues. La disponibilité des données n'est pas essentielle. L'utilisateur est contraint d'arrêter son travail. Les données peuvent être perdues ou altérées.
AEC-1	Hautement fiable	Les fonctions peuvent être interrompues, pour autant que la disponibilité des données soit assurée. L'utilisateur est contraint d'arrêter son travail. Cependant une sauvegarde des données existe, ainsi qu'un journal de bord pouvant servir à identifier et à récupérer les transactions incomplètes.
AEC-2	Haute disponibilité	Les fonctions peuvent subir de légères interruptions de service durant le temps de travail. L'utilisateur peut être interrompu pendant un temps assez court. Cependant, des transactions peuvent être relancées à partir du journal de bord, et les performances peuvent subir une certaine dégradation.
AEC-3	Résilient	Les fonctions demandent un traitement sans interruption durant le temps de travail. L'utilisateur reste en ligne. Cependant, une transaction peut être relancée à partir du journal de bord, et les performances peuvent subir une légère dégradation.
AEC-4	Tolérant	Les fonctions demandent un traitement continu, toute panne étant transparente pour l'utilisateur. Aucune interruption de travail, aucune transaction perdue, aucune dégradation des performances. Les fonctions doivent être opérationnelles 24 heures sur 24, et 7 jours sur 7.
AEC-5	Tolérant à une catastrophe	Les fonctions doivent être disponibles en toute circonstance.

Bibliographie

[DPI00] D. L. PIPKIN, 2000, Sécurité des systèmes d'information, Edition CampusPress, Paris, 391p

[INF06] INFOLEARN, 2006, *Les fondamentaux de l'IT Infrastructure Library*, INFOLEARN, Genève, 96p

[JAN08] O. JAN, 2008, *NAGIOS au cœur de la supervision Open Source*, Editions ENI, St-Herblain, 375p

[FRE08] Ph. FREDDI, 2008, *Mise en œuvre de systèmes hautement disponibles* IN *Windows Server 2008*, Editions ENI, St-Herblain, p510 à p521

[NOI08] C. NOIRAULT, 2008, *ITIL V3 Les meilleures pratiques de gestion d'un Service Informatique*, Edition ENI, St-Herblain, 274p

[SBG08] M. SCHUBERT, D. BENNETT, J. GINES, A. HAY, J. STRAND, 2008, *Nagios 3 Entreprise Network Monitoring Including Plug-Ins and Hardware Devices*, Syngress Publishing, Burlington, 348p

[IPL10] IP-LABEL.COM, 2010, Pour *une performance durable du cloud computing*, Livre blanc, Ip-Label.com, 60p

[ARG10] Toute la documentation du projet ARGOS, projet de surveillance à l'État de Genève. L'appel d'offres, ainsi que les réponses des éditeurs. Quatre conférences de présentation de solutions.

[WWW01] http://www.bmc.com, 3.07.2010

[WWW02] http://www.commentcamarche.net, 17.06.2010

[WWW03] http://www.economypoint.org/h/high-availability.html, 18.08.2010

[WWW04] http://www.itilfrance.com/, 17.04.2010

[WWW05] http://www.journaldunet.com/solutions/dossiers/pratique/supervision.shtml 20.03.2010

[WWW06] http://www.journaldunet.com/solutions/expert/exploitation/42797/le-business-service-management--un-skms-avant-la-lettre.shtml, 02.09.2010

[WWW07] http://www.linux.com/feature/60644, 6.07.2010

[WWW08] http://www.luffy.cx/zabbix.pdf, 12.07.2010

[WWW09] http://msdn.microsoft.com/fr-fr/library/aa291543%28VS.71%29.aspx, 21.06.2010

[WWW10] http://www.microsoft.com/systemcenter/en/us/operations-manager.aspx, 28.07.2010

[WWW11] http://wiki.monitoring-fr.org/, 19.12.2009

[WWW12] http://www.nagios.org, 19.12.2009

[WWW13] http://www.nagvis.org/, 09.05.2010

[WWW14] http://www.nconf.org/, 09.09.2010

[WWW15] http://www.newsitweb.info, 14.05.2010

[WWW16] http://www.oracle.com/technetwork/oem/extensions/index.html, 02.09.2010

[WWW17] http://doc.pcsoft.fr/fr-FR/?3541100, 03.09.2010

[WWW18] http://www.pingdom.com/, 23.06.2010

[WWW19] http://www.sap.com/france/platform/netweaver/components/solutionmanager/ index.epx, 02.09.2010

[WWW20] http://www.socard.fr/fiabilite.htm, 21.06.2010

[WWW21] http://raisin.u-bordeaux.fr/IMG/pdf/zabbix.pdf, 6.07.2010

[WWW22] http://www-igm.univ-mlv.fr/~dr/XPOSE2006/JEREMIE_LEGRAND_HAUTE_DOSPO/theorie4.htm, 17.08.2010

[WWW23] http://www.webinject.org, 12.07.2010

[WWW24] http://www.zabbix.com, 8.07.2010

Liste des figures

Liste des tableaux

Résumé

Mots clé : Surveillance, système d'information, Nagios, haut disponibilité, qualité, continuité de service, gestion des incidents.

Ce mémoire traite de l'extension de la surveillance du système d'information. L'objectif est d'être en mesure de fournir à l'utilisateur final un état des lieux de la disponibilité de ses outils de travail.

Au travers d'une étude de faisabilité et d'une preuve de concept réalisé dans un environnement hétérogène existant, ce mémoire apporte des réponses aux principales difficultés rencontrées. Ces obstacles sont, premièrement, l'augmentation de la complexité au niveau de la gestion des accès aux données de disponibilité publiées et, deuxièmement, le manque de maîtrise des retombées provoquées par la mise à disposition de l'utilisateur des informations de disponibilité.

La première étape indispensable à la réalisation de cette démarche est d'inventorier les données de surveillance déjà existantes, tout en définissant les rôles et responsabilités de chacun.

L'approche présentée est structurée en quatre parties: une définition du sujet, une analyse détaillée de la problématique de la surveillance d'un système d'information, un modèle conceptuel d'une méta-surveillance d'un système d'information qui s'appuie sur les référentiels existants de l'entreprise, et pour terminer une preuve du concept pour démontrer l'application du modèle à l'infrastructure de l'entreprise, en tenant compte des contraintes imposées.

Abstract

Keywords: Monitoring, information system, Nagios, high availability, quality, business continuity, Incident management.

The subject of this engineer thesis covers the extension of monitoring information system. The goal is to be able to provide the end-user an overview of the availability of its tools.

Through a feasibility study and a proof of concept performed in an existing heterogeneous environment, this paper provides answers to this challenge. The difficulties are, first, the increased complexity in the management of access to availability data published and, secondly, the lack of control on the impacts caused by the fact of providing users with availability information.

The first essential step towards the realization of this approach is to inventory existing monitoring data, and defining everyone's roles and duty.

This thesis is composed of four principal chapters: a definition of the subject, a detailed analysis of the problematic of monitoring an information system, a conceptual model of a meta-monitoring of information system that relies on information with existent repository in the company, and last, a proof of concept to attest the application of the model in the enterprise infrastructure, with the existing constraints.