Aspects fonctionnels et techniques de la supervision d'un système d'information

Timothée Foucart

CONSERVATOIRE NATIONAL DES ARTS ET MÉTIERS
CENTRE RÉGIONAL ASSOCIÉ DE BELFORT

Mémoire
présenté en vue d'obtenir
LE DIPLÔME D'INGÉNIEUR CNAM

Spécialité : INFORMATIQUE
OPTION : systèmes d'information

par
Timothée Foucart

---

aspects fonctionnels et techniques de la supervision d'un système d'information
études et mise en œuvre de procédés de supervision

JURY

PRÉSIDENT :	Monsieur Kamel Barkaoui
MEMBRES :	Monsieur A.-J. Fougères
	Monsieur P. Descamps
	Monsieur A. Schili
TUTEURS :	Monsieur Y. Daouze
	Monsieur R. Baudiquey

Remerciements

Je tiens à remercier ici toutes les personnes qui, de près ou de loin, m'ont aidé et soutenu durant la mise en œuvre de ce projet et l'écriture de ce mémoire.

Ces remerciements s'adressent plus particulièrement à :

- mes deux collègues, Renaud Baudiquey et Nelly Guyerdet avec qui nous formons le pôle 'système d'information' de Grand Besançon Habitat. Je suis spécialement reconnaissant envers Renaud Baudiquey, qui a accepté de tenir le rôle de tuteur conjointement à Yves Daouze.

- Yves Daouze, directeur général de GBH, qui a bien voulu assumer le rôle de tuteur en cours de projet, suite au départ de Rémy Heintzmann.

- Catherine Berthet, pour son fastidieux travail de correction, ses exigences typographiques qui dépassent de loin les miennes et sa patience.

- Claire et les filles, mes plus fidèles alliées, qui se demandent souvent s'il existe une vie après le CNAM.

- Armand Guy, pour son indéfectible et précieux soutien.

Je tiens enfin à saluer Alain-Jérôme Fougères pour les nombreux conseils et le temps qu'il a pu me consacrer lors de la rédaction de ce mémoire.

Liste des abréviations

API	*application programming interface* (interface de programmation)
APM	*application performance management*
CIM	*common information model*
DMZ	*demilitarized zone*
DNS	*domain name system*
EUE	*end user experience* : tests comportementaux
ERP	*enterprise ressource planning*
ETL	*extract transform load*
GBH	Grand Besançon Habitat (office public de l'habitat de Besançon)
GIE	groupement d'intérêt économique
IAB	*Internet Activities Board*
IHM	interface homme-machine
ITIL	*Information Technology Infrastructure Library*
IPMI	*intelligent platform management interface*
KPI	*key performance indicator*
KVM	*keybord-video-mouse switch*
LDAP	*lightweight directory access protocol*
LTO	*linear tape open*
LED	*light-emitting diode*
NAS	*network attached storage*
NRPE	*Nagios remote plug-in executor*
NSCA	*Nagios service check acceptor*
OID	*object identifier*
OPH	office public de l'habitat
SAN	*stockage area network*
SEPA	*Single European Payment Area*
SI	système d'information
SLA	*service level agrement*
SNMP	*simple network management protocol*
SMI	*storage management initiative*
SMTP	*service message transport protocol*
SSII	société de services en industrie informatique
WBEM	*web based enterprise management*

Tableau 1 : liste des abréviations

Table des matières

Introduction

Dans un système d'information, la question de la performance et de sa mesure est une préoccupation de fond dont les motivations se déclinent souvent au niveau global, impliquant l'organisation dans son ensemble. On peut par exemple y associer des objectifs financiers, opérationnels ou stratégiques dont la portée concerne davantage un pilotage directif[1] que les praticiens du système d'information. À partir de là, deux perspectives peuvent être distinguées :

- celle du pilotage du SI, qui valorise une approche *top-down* et considérerait le système d'information comme une entité définie dont l'activité est quantifiable ;
- celle des praticiens mêmes du système d'information, à savoir l'ensemble de ses utilisateurs, où l'on pourrait considérer le système d'information comme un ensemble flou construit de pratiques autour d'outils.

Ces deux perspectives, non qu'elles s'opposent nécessairement[1], témoignent de la complexité de la nature d'un système d'information[2], mais également des liens ténus qu'il entretient avec l'organisation. La question de l'évaluation du système d'information renvoie à cette dualité, mais tend à limiter sa portée à la seule préoccupation de l'équipement informatique et de son fonctionnement. Cette vision limitative explique que dans une organisation, le système d'information soit le fait des informaticiens.

Il reste toutefois délicat d'observer une séparation stricte entre les pratiques collectives ou individuelles autour de l'outil informatique et la technique informatique, simplement parce que le système d'information concerne, par nature, tous ses usagers. Aussi, la tentation de distinguer le « technique » du « fonctionnel » comme s'il s'agissait de deux domaines indépendants pose la question du rôle de l'outil informatique au regard de l'organisation.

La véritable difficulté provient sans doute de la perception de l'outil que l'on se construit. Par exemple, lors de demandes d'assistance des usagers au service informatique, on constate fréquemment que ce qui «fonctionne» ou ce qui «ne marche pas» n'est pas nécessairement le fait d'un dysfonctionnement technique, mais peut tout aussi bien être l'effet d'une pratique inappropriée, l'expression d'un ressenti, d'une croyance ou

1 Une approche critique à propos de la théorie de la pratique pourrait être celle de P. Bourdieu[2] qui oppose une vision théorique à sa *praxéologie*. En ce sens, on pourrait considérer le système d'information comme un ensemble construit de pratiques, plutôt que comme une entité dont on présuppose la cohérence et que l'on évalue par des indicateurs globaux.

2 Une définition du système d'information : « Un système d'information (SI) est un ensemble organisé de ressources (matériels, logiciels, personnel, données et procédures) qui permet de collecter, regrouper, classifier, traiter et diffuser de l'information dans un environnement donné »[3]

d'une humeur. Ce renvoi systématique à la représentation que l'on se fait de l'outil et à la perception d'un dysfonctionnement plutôt qu'à sa fonction idéale et théorique met en exergue les limites d'une approche *technocentrique[4]* qui associe le SI à la seule infrastructure matérielle. Parallèlement, ceci explique la difficulté pour les gestionnaires du SI à observer cette forme de neutralité que l'on appose aux outils qu'ils gèrent. Ainsi, on pourrait opposer un déterminisme technologique[3], où l'on considérerait la technique comme principal facteur d'influence de l'organisation, à un questionnement plus systémique où il s'agirait d'évaluer « pourquoi, comment et dans quelles limites les caractéristiques d'une technologie deviennent contraignantes pour les acteurs et comment jouer avec ces contraintes »[6]. Or, c'est dans cette ambivalence que s'inscrit l'activité d'un SI, la liant d'un côté au complexe organisationnel et aux comportement de ses acteurs et de l'autre à son domaine d'intervention privilégié : le matériel et son fonctionnement.

Intuitivement, ceci explique la difficulté à mettre en évidence un lien entre la performance du SI et celle de l'organisation[5]. Pourtant, l'articulation des SI avec les besoins des métiers est une exigence très largement répandue et affirmée[8]. Cette dernière pose évidemment la question de l'évaluation et de la mesure, sans pour autant en indiquer les conditions : concrètement, une solution technique qui démontre des capacités de performance brutes (latence, débit, nombre d'opérations par seconde, etc.) ne suppose pas pour autant que l'usage qui en est fait est conséquemment performant. À l'opposé, il reste particulièrement complexe pour un utilisateur de mesurer l'écart entre le dysfonctionnement qu'il perçoit ou ressent et celui que l'on peut effectivement imputer aux capacités techniques de l'outil qu'il utilise.

À cela s'ajoute la complexité intrinsèque du système d'information. L'architecture matérielle, applicative et fonctionnelle possède en effet ses propres particularités, tant par les mécaniques mises en œuvre que par les moyens de les articuler en un *système,* c'est-à-dire en *un ensemble d'éléments interagissant selon certains principes ou règles[9].* Ce socle technique, qui reste le plus souvent transparent pour l'utilisateur, est pourtant d'une importance capitale, puisqu'il assure que les outils mis en œuvre soient disponibles.

La question consiste donc à savoir comment évaluer le système d'information en prenant en compte les exigences techniques du matériel, celles de l'organisation et celles des usagers du SI. Comment évaluer l'écart entre ce qui « ne marche pas » du point de vue d'un utilisateur et un problème technique ? En quoi cela impacte-t-il les objectifs globaux de l'organisation ? Ce questionnement interroge non seulement le fonctionnement du SI, qu'il s'agisse de ses performances, de ses défaillances ou de sa disponibilité,

3 Le déterminisme technologique peut se définir de la façon suivante : « On suppose que le changement technique est un facteur indépendant de la société. D'une part, le changement technique est autonome [...]. D'autre part, un changement technique provoque un changement social. » Vinck, p 304 voir [5]

mais également les limites de son champ d'action.

Dans cette optique, la mise en œuvre d'un système de supervision est apparue comme un moyen évident d'appréhender le SI dans sa globalité, c'est-à-dire non seulement du point de vue technique, mais également comme un système évolutif soumis aux exigences des utilisateurs et à celles de l'organisation. L'idée de « superviser » peut s'entendre comme une action visant à contrôler ou à vérifier. Étymologiquement, on peut décomposer le mot en « super » : *au-dessus* et « videre » : *observer, prendre des mesures pour, discerner sans nécessairement y prendre attention*[4]. Du point de vue d'un système d'information, la supervision repose sur des outils qui automatisent cette logique de contrôle en s'appuyant sur des procédés d'automatisation et sur la métrologie. Au-delà d'une simple « surveillance » automatisée, on peut se demander comment un tel outil peut rendre compte de l'activité du SI, c'est-à-dire sans le confiner à ses aspects mécaniques ? Concrètement, comment les techniques de supervision sont-elles en mesure d'évaluer la capacité d'un SI à satisfaire les objectifs fonctionnels d'une organisation ?

Ce questionnement, qui incorpore le système d'information comme une composante de l'organisation à part entière, a été largement présent dans la mise en œuvre d'un projet de supervision à Grand Besançon Habitat. De façon très pragmatique, il s'agit de vérifier comment les techniques de supervision permettent de contrôler non seulement la dimension technique du SI, mais également d'en éprouver la fonction même, en s'appuyant sur des données de métrologie et des procédés d'évaluation du ressenti utilisateur[5].

Le projet, qui s'est organisé en plusieurs étapes, consiste en la mise en place d'un processus d'évaluation capable d'évoluer avec le SI. Il inclus une démarche exploratoire dont certains aspects expérimentaux ont dû être écartés progressivement, soit en raison de leur manque de stabilité, soit en raison de leur manque de pertinence.

Sur le fond, il s'agit de vérifier comment des outils de supervision peuvent s'organiser en une solution cohérente capable non seulement de restituer l'état technique du SI, mais également d'en évaluer la pertinence sur le plan stratégique.

C'est dans cette perspective qu'il a semblé intéressant, dans un premier temps, de s'appuyer sur le référentiel ITIL. Ce dernier promeut l'alignement stratégique du système d'information aux besoins métier et permet de dégager ce que peuvent être, au sens large, les enjeux de la supervision. Si ces préconisations ITIL permettent de dégager un certain nombre de concepts, elles doivent toutefois être confrontées à deux réalités : d'un côté celle du SI de GBH qui possède ses propres besoins et spécificités et de l'autre

4 A. Dauzat, J. Dubois et H. Mitterand, *Nouveau dictionnaire étymologique*. Paris : France loisirs, 1980.

5 *End User Experience* (EUE) : technique visant à évaluer des simulations de comportement d'un utilisateur.

celle des différents modèles de gestion et standards comme WBEM ou SNMP sur lesquels reposent les nombreuses implémentations d'outils de supervision existant sur le marché. Ce sont ces points que nous observerons dans un second temps. D'une part en restituant un exemple concret d'analyse où l'on tente d'évaluer les cibles effectives d'un procédé de supervision ; d'autre part en observant les fondements théoriques des normes et standards de supervision les plus courants.

C'est en confrontant préconisations (ITIL), normes (SNMP, CIM, WBEM, etc.) et réalité pratique du SI de GBH que nous motiverons le choix d'une solution de supervision. Bien qu'un grand nombre de ces solutions soient présentes sur le marché, toutes n'autorisent pas la même souplesse d'adaptation. Or, le SI de GBH est, de fait, constitué de composants particulièrement hétérogènes. C'est principalement pour faciliter la prise en compte des particularismes et la maîtrise des procédés de supervision que le choix s'est arrêté sur la mise en place de deux solutions complémentaires :

- la première, dénommée *Shinken*, est une solution de supervision sous licence libre qui repose sur un modèle hérité de *Nagios* , solution elle-même particulièrement répandue et populaire. C'est en examinant les différentes caractéristiques de ce produit et ses principes fondateurs que nous vérifierons sa capacité à s'adapter aux modèles examinés précédemment ;
- la seconde, *Canopsis*, est une solution open-source dite d'*hypervision.* Elle s'attache davantage aux questions de la collecte des données de supervision, à la composition d'indicateurs complexes et à leur représentation graphique. *Canopsis* valorise également une approche alternative, de type APM[6], dont nous éprouverons la pertinence.

Outre l'architecture et les modalités de communication entre ces deux plates-formes, nous observerons comment le traitement des données de supervision peut concourir à fournir des éléments capables d'améliorer la lisibilité du SI pour ses utilisateurs finaux et conjointement, produire des indicateurs synthétiques capables d'évaluer l'alignement stratégique du SI.

Une dernière partie s'attachera enfin à relever les aspects opératoires de la mise en œuvre effective de ce projet. De la configuration de l'environnement technique à la programmation de sondes et à la construction de tableaux de bord, les solutions adoptées démontrent une certaine efficacité. Leur mise en œuvre recèle toutefois une certaine complexité, et parfois quelques fragilités, ainsi que nous l'observerons au travers des résultats produits.

6 L'APM, ou *Application Performance Monitoring* regroupe des procédés techniques qui visent à évaluer les performances applicatives en s'appuyant sur l'usage plutôt que l'infrastructure. Voir [10]

1.Contexte

L'office public de l'habitat de Besançon, également connu sous la dénomination « Grand Besançon Habitat » (GBH) est un bailleur social qui gère un parc d'environ 5500 logements sur la communauté de communes de la ville de Besançon.

À l'origine, Grand Besançon Habitat était un établissement public à caractère administratif, dénommé OPHLM. Ces offices ont été transformés en offices publics de l'habitat par une ordonnance de février 2007. Ce sont des Établissements Publics à caractère Industriel et Commercial (EPIC) : ils disposent de leurs fonds propres mais restent astreints à certaines règles, comme celles de la comptabilité publique ou du code des marchés publics.

GBH dispose d'un patrimoine d'immeubles destinés à la location. Les particuliers qui répondent à certaines conditions de ressources peuvent y accéder. GBH est non seulement chargé d'assumer les tâches administratives inhérentes à un bailleur, mais également d'assurer la maintenance, la rénovation, la construction ou la destruction de son parc.

GBH est constitué de trois agences sur Besançon même et son siège est situé dans le quartier Planoise. Environ 150 salariés y travaillent. L'activité métier se répartit entre différents pôles ou directions - ainsi que détaillé dans l'Annexe 6.1 : la gestion locative (qui comprend les agences réparties dans Besançon), le pôle finances et comptabilité, le pôle développement - construction et maintenance, les ressources humaines et l'informatique.

Le service informatique est un service transversal qui a été détaché du pôle finances et comptabilité en 2013, puis rattaché à la direction générale. Il est chargé d'assurer, au sens large, la gestion de l'infrastructure informatique. L'activité métier de GBH s'appuie en effet sur des solutions informatisées complexes. On dénombre 70 serveurs virtualisés et une quinzaine de serveurs physiques en production. En outre, un certain nombre de services, comme la téléphonie, l'impression, les accès réseau, la production de statistiques ou le calcul des factures et des charges, nécessitent de disposer d'une infrastructure matérielle et logicielle spécifique. Il revient au service informatique, qui dispose d'un budget propre, d'assurer le bon fonctionnement de l'infrastructure logique et technique.

Bien que ses domaines d'intervention puissent sembler diffus, les activités du service informatique de GBH correspondent assez exactement à la description qu'en fait ITIL et dont le schéma est représenté par l'Illustration 1.1.

Illustration 1.1 : les activités du SI selon ITIL

Le service informatique est supervisé par le directeur général, Yves Daouze. Il est composé de trois personnes : deux administrateurs système et réseau et une assistante fonctionnelle qui est chargée de faire le lien entre le fonctionnel - c'est-à-dire tout ce qui relève spécifiquement du métier - et la technique informatique, qui relève des compétences des deux administrateurs, dont je fais partie.

Les activités du service se répartissent globalement entre l'assistance aux utilisateurs, la gestion des projets et des transitions et les opérations de maintenance et d'évolution de l'infrastructure. Dans ce contexte, très peu de services sont sous-traités ou externalisés ; ceci explique que l'intégralité de l'administration système et réseau revienne directement au service informatique. Le panel d'activités est donc extrêmement riche et varié, d'autant qu'il induit la maîtrise de systèmes et composants hétérogènes.

L'infrastructure serveur repose essentiellement sur un principe de virtualisation avec des hyperviseurs de type *ESXi* ou *XenServer*. La baie serveur, située au siège, inclut donc un *IBM BladeCenter H* doté de 7 serveurs lames, deux baies SAN fibre optique et quelques serveurs physiques, dont un *IBM Power i7* tenu par un système OS/400. Les liens réseau entre le siège et les agences reposent sur un réseau de fibre optique partagé ou privé sur certains liens : le Réseau Lumière[7]. Le périmètre d'interven-

7 Voir : http://www.besancon.fr/index.php?p=511&art_id=1251 : « Le Réseau Lumière est constitué d'anneaux interconnectés et d'antennes d'optiques. Avec 84 km de câble transportant plus de 1700 km de fibres optiques, il dessert actuellement 110 sites appartenant à 9 administrations. [...] Du point de

tion du service informatique recouvre donc tous les éléments réseau actifs, la baie serveur, les copieurs multifonctions, la téléphonie IP, ainsi que l'ensemble des postes utilisateurs. On dénombre une centaine d'utilisateurs répartis au siège ou dans les agences (Illustration 1.2).

Octroyant une certaine autonomie à l'office public de l'habitat, cet état de fait nécessite toutefois une certaine polyvalence puisque les plates-formes, systèmes et composants applicatifs se multiplient. Depuis quelques années, le système d'information de GBH ne cesse d'évoluer et de s'étoffer au gré des réformes légales, des besoins et des changements technologiques.

La question de la supervision s'est donc affirmée d'une part comme un moyen d'évaluer la qualité générale du système d'information et d'autre part comme un outil de gestion centralisé capable de fournir une aide à l'identification des problèmes et à leur résolution. En dernier lieu, il s'agit également de pouvoir rendre compte de l'activité du SI et de la valoriser comme une composante active de l'organisation.

Ce projet a été mené parmi d'autres au cours des années 2013 et 2014 : réforme de la demande de logement[8], mise en place du SEPA[9], installation d'un SAN fibre optique, renouvellement des postes de travail, migration des serveurs Citrix, etc.

De fait, il reste délicat de valoriser ce type de projet :

- en premier lieu parce que l'organisation de GBH a été transformée cette même année ;
- ensuite parce que le caractère critique de la supervision peut sembler discutable au regard d'autres projets dont les finalités sont sans doute plus en lien avec l'activité métier.

vue administratif, ce réseau indépendant à usage partagé, qui contribue par sa vocation à l'aménagement du territoire, bénéficie d'une autorisation (Groupe Fermé d'Usagers – GFU), émise par l'A.R.T. Les partenaires sont liés par une convention de copropriété et d'entretien et d'usage ». GBH est membre du GFU.

8 La gestion de la demande de logement a récemment fait l'objet d'une réforme dans le département du Doubs : voir [11]

9 La mise en œuvre du SEPA, ou *Single European Payment Area* a conduit à d'importantes modifications dans les processus de paiement [12]

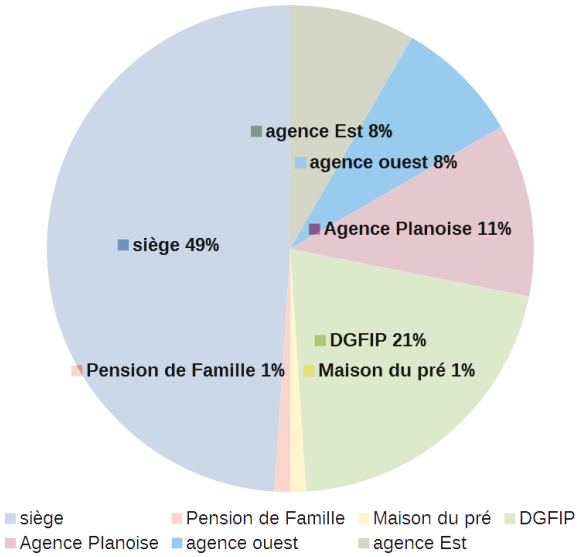


Illustration 1.2 : répartition des utilisateurs du SI de GBH par site

En ce qui concerne la supervision, on dispose en effet d'emblée d'une multitude d'outils disparates et limités au champ d'un matériel ou d'une application. Si l'intention d'unifier la supervision semble à priori pertinente, elle nécessite toutefois d'être démontrée, tant elle repose sur des abstractions, des concepts et des mécanismes techniques complexes. Cette complexité est le symptôme de la difficulté de représenter le SI de façon exhaustive et globale à partir d'un seul point de vue. Ceci explique que le projet ait avant tout été le fait du service informatique, dont la transversalité offre un point de vue privilégié sur l'organisation dans son ensemble. La véritable difficulté a consisté à tenter de dépasser le seul point de vue technique, à le confronter à la réalité des besoins de GBH et à mettre en place des outils et des processus qui permettent d'abstraire les composants du SI de leurs seules caractéristiques techniques.

2. Enjeux et architectures

Dans ce chapitre, nous reprenons les éléments qui ont motivé la mise en œuvre d'un système de supervision. Le référentiel ITIL[13] constitue à ce niveau une référence pertinente pour justifier l'intérêt du projet. L'exploration rapide de quelques modules du référentiel permet de mettre en évidence les enjeux attenants.

L'approche ITIL fournit un socle relativement pertinent quant à ce que pourrait être la supervision du SI au regard de l'organisation. À partir de là, nous observerons, à travers un exemple concret, comment envisager l'analyse de l'architecture du SI en partant d'enjeux « métier ». Cette démarche analytique ne peut néanmoins se soustraire aux outils existants - ou plus exactement, aux standards, normes et modèles et à leurs implémentations. Les outils que l'on peut solliciter dans le cadre de la supervision d'un SI reposent en effet sur des protocoles de gestion dont les tenants et les aboutissants sont eux aussi liés à des représentations ou modèles de ce qu'est un système d'information.

Enfin, nous aborderons la question du choix d'une solution de supervision en essayant d'articuler la réalité du système d'information avec ses enjeux stratégiques effectifs et les moyens techniques disponibles.

2.1. ITIL et les enjeux de la supervision

ITIL est l'acronyme d'*Information Technology Infrastructure Library.* ITIL consiste en un cadre de référence rassemblant, dans un ensemble de guides, les meilleures pratiques en matière de management des services informatiques. Son objectif est d'améliorer le service rendu par les directions informatiques, afin d'apporter une meilleure qualité de service aux utilisateurs. La supervision du SI participe à cette recherche d'amélioration, de même qu'un référentiel comme ITIL.

La bibliothèque ITIL a été initialisée en 1989 par le CCTA (*Central Computing & Telecommunication Agency*) du gouvernement du Royaume-Uni, puis pilotée par l'OGC (*Office of Government Commerce*) qui remplace le CCTA. Dans sa troisième version, ITIL a fait intervenir les experts de plusieurs SSII. L'objectif d'ITIL est de doter les directions des systèmes informatiques (DSI) d'outils et de documents leur permettant d'améliorer la qualité de leurs prestations, c'est-à-dire améliorer la satisfaction de leurs clients tout en répondant au mieux aux objectifs stratégiques de l'organisation dans laquelle elles s'inscrivent.

Pour ce faire, l'approche consiste à considérer le service informatique comme un en-

semble de processus étroitement liés. Pragmatiquement, ITIL répond à la logique visant à faire en sorte que l'informatique soit au service du personnel et des clients et non l'inverse. La démarche ITIL n'a pas comme seuls bénéficiaires les directions informatiques, puisqu'elle consiste à sensibiliser ces dernières au fait que la qualité et la disponibilité de l'infrastructure technologique ont un impact direct sur la qualité globale de l'entreprise. Dans cette visée, le projet de supervision correspond à cette recherche globale de qualité, tout en valorisant une logique de service décentrée de l'objet technologique. En effet, ITIL décrit les différentes activités sous forme de services :

> ***un service*** *est un moyen de délivrer de la valeur aux clients en facilitant la production des résultats dans leurs activités sans qu'ils aient à se préoccuper des coûts et des risques spécifiques au service qui leur est fourni.[14]*

Définie de cette façon, la notion de service reste très générique. Le vocabulaire ITIL, économique et commercial (« production », « valeur », « résultat », « coût », etc.) peut parfois sembler confus lorsque l'on cherche à l'appliquer à un domaine dont les motivations ne relèvent pas directement de ces logiques. Du point de vue de l'administration systèmes et réseaux du SI, en identifiant les utilisateurs du SI comme des « clients », les services pourraient correspondre à tous les moyens informatiques mis en œuvre en vue d'automatiser, au moins partiellement, les tâches liées aux processus métier auxquels ils participent.

Par exemple, une application capable de générer automatiquement la structure d'un appel d'offres peut être perçue comme un « service »: ce qui intéresse le « client », c'est effectivement le résultat - dans ce cas, il s'agit de l'élaboration de son appel d'offres et non des procédés techniques qui permettent d'y parvenir. Les ressources mises en œuvre dans un SI ne sont donc pas nécessairement visibles pour l'utilisateur final. Typiquement, les services sont systématiquement liés à des capacités : volume, bande passante, branchements disponibles, etc. Dans les procédés de supervision, ces capacités sont l'objet de la métrologie :

> ***les capacités*** *correspondent aux processus et fonctions utilisés pour gérer des services. Les capacités sont des ressources intangibles d'une organisation et ne peuvent pas être vendues mais peuvent se développer et gagner ainsi en maturité au fil du temps.[15]*

La *gestion de la capacité* revient à s'assurer que les services puissent être disponibles au moment adéquat. De façon très claire, la maintenance, qu'elle soit préventive ou curative, participe à la gestion des capacités. Dans cette optique, un outil de supervision qui facilite les opérations de maintenance préventive est un outil de gestion des capacités. ITIL distingue trois sous-processus dans la gestion des capacités[16] :

- la gestion de capacité des ressources (RCM) qui concerne plus directement l'éva-

luation des composants en termes de performance ;

- la gestion de la capacité des services (SCM) qui permet de gérer, contrôler et prévoir la charge des différents systèmes informatiques ;
- la gestion de la capacité *business* (BCM) relève de la gestion des plans stratégiques au niveau commercial. Cet aspect concerne moins directement la supervision de l'infrastructure.

Cette gestion des capacités met en avant l'idée que l'activité de l'organisation, quelle que soit sa connotation, repose sur des services. Ces derniers s'appuient sur des systèmes et des composants dont l'évaluation en termes de performance est indissociable de l'activité organisationnelle.

Globalement, ITIL met en avant l'idée que l'organisation d'une entreprise peut s'améliorer dès lors que l'on identifie les processus qui y sont mis en œuvre :

> ***un processus*** *est un ensemble d'activités coordonnées mettant en œuvre des ressources et des capacités en vue de produire un résultat aux clients.[15]*

Le chapitre d'ITIL portant sur la conception de service[15] explique un certain nombre de règles et de bonnes pratiques à prendre en compte lors de la conception d'un service. La conception d'un service est une chose complexe du fait de l'existence du grand nombre de paramètres à considérer : acteurs, points de vue et objectifs. En vue de garantir le niveau de service fourni, ITIL propose de fixer des accords sur les niveaux de service *(Service Level Agrement* ou SLA*)* :

> ***un accord sur les niveaux de service*** *est un accord écrit entre un fournisseur de services et un ou des clients. Il porte sur un ou plusieurs services d'affaires et décrit les niveaux de services prévus avec la ou les organisations d'affaires (disponibilité, capacité, sécurité et continuité de service).[14]*

Plus loin, ITIL propose d'associer à ces SLA des indicateurs-clés de performance :

> ***un indicateur-clé de performance*** *(Key Performance Indicator ou KPI) est un ensemble de métriques objectives et mesurables qui permettent d'évaluer si les différents niveaux de services convenus (SLA) ont été respectés.[14]*

Ces indicateurs servent notamment à définir des pénalités lorsque les niveaux de services n'ont pas été respectés par les parties.

Les préconisations ITIL relatives à la conception de services mettent en avant la notion d'évaluation et de mesure, sans pour autant préconiser les moyens de mise en œuvre de cette évaluation. Au travers de la gestion des capacités émerge la notion de disponibilité des services. Plus loin, l'évaluation de cette disponibilité est précisée avec la définition des SLA et KPI. Ces indicateurs synthétiques sont un des objectifs de la su-

pervision ; ils fournissent une modalité d'évaluation des services.

Pour autant, ces notions qui concernent plus directement la phase de conception de services sont indissociables de la question de leur exploitation. Cette dernière est traitée dans le chapitre ITIL *Service Operation[17].*

L'exploitation des services correspond au moment où les services sont en production. Théoriquement, les accords sur les SLA sont en place mais l'activité n'est pas figée : il faut pouvoir anticiper les événements, les changements et besoins futurs. La gestion des événements oppose la réactivité, qui consiste à réagir une fois qu'un événement survient, à la proactivité, qui consiste à anticiper celui-ci. Les événements se définissent comme suit :

> ***un événement*** *est une occurrence détectable ou discernable ayant une signification sur la gestion d'une infrastructure ou la fourniture d'un service et une évaluation de l'impact indiquant qu'une déviation pourrait apparaître sur les services.[14]*

La notion d'*impact* évoquée dans cette définition n'est pas neutre : elle suppose des liens d'interdépendance entre les composants d'un service et rappelle l'idée que l'on intervient dans un système. Les événements, quant à eux, peuvent revêtir des formes bien différentes. ITIL les classe en trois catégories :

- les événements *normaux* : connexion effectuée avec succès, temps de réponse acceptable ;
- les événements *inhabituels* : avertissement lors de l'exécution d'un programme, activité plus forte que la moyenne, élévation des temps de réponse ;
- les événements signalant une *exception* : connexion impossible, perte d'un disque, etc.

Ces événements peuvent être les symptômes d'incidents ou de problèmes. ITIL distingue nettement ces deux notions :

> ***un incident*** *est un événement qui ne fait pas partie du fonctionnement standard d'un service et qui peut entraîner une interruption ou une baisse de qualité pour un service.[14]*

Les événements ayant un impact potentiel mais non encore observé sont également considérés comme des incidents. Un incident peut donc être éradiqué sans investigation supplémentaire, par exemple avec une réparation immédiate ou un contournement[14]. Le fonctionnement « normal » d'un service s'entend, au sens d'ITIL, comme celui défini dans les contrats de SLA. ITIL distingue les problèmes des incidents :

un problème *est la cause d'un ou plusieurs incidents. Lors de l'enregistrement d'un problème, la cause exacte n'est pas toujours connue. On considère qu'un problème est le symptôme d'une erreur inconnue dans l'infrastructure[14].*

Dans le processus ITIL de gestion des incidents, les outils de supervision occupent une place importante. Ils doivent concourir à :

- détecter et enregistrer l'incident ;
- classifier l'incident ;
- isoler, examiner et diagnostiquer.

Enfin, dans le module *Continual service improvement* (l'amélioration continue des services)[18], ITIL rappelle trois règles de base, celles-là mêmes qui permettent « d'aligner et de réaligner en permanence les services informatiques sur les besoins d'affaires en perpétuelle évolution, ceci en identifiant et en implantant les améliorations sur les services informatiques supportant les processus d'affaire » :

- vous ne pouvez pas gérer ce que vous ne pouvez pas contrôler ;
- vous ne pouvez pas contrôler ce que vous ne pouvez pas mesurer ;
- vous ne pouvez pas mesurer ce que vous n'avez pas défini.

Ces quelques éléments tirés des préconisations ITIL permettent d'établir des repères fondamentaux pour cerner le rôle d'un outil de supervision. Si l'on reprend de façon synthétique ces quelques éléments tirés de divers chapitres d'ITIL :

- les activités de l'entreprise sont identifiées de façon générique comme des services ;
- ces services sont en mesure d'attendre un certain niveau de qualité. Cette dernière repose sur des processus de gestion des ressources et des capacités, qui induisent donc une évaluation et des mesures ; c'est à ce niveau que la métrologie devient incontournable. Les outils de supervision peuvent en effet participer à ces processus : surveillance des consommations de CPU, de l'occupation d'un volume de disque par exemple. Par extension, c'est à partir de ces mesures que l'on peut calculer des KPI et vérifier l'ajustement au SLA ;
- les procédés de supervision participent également au processus de gestion des incidents, que ce soit comme moyen d'analyse, d'enregistrement ou de détection. Il convient en effet que les outils de supervision soient en mesure de distinguer si un événement est « normal » ou non ; dans quel cas il doit être en mesure de déclencher une alerte et de la prioriser. Ce qui importe, c'est que le service puisse fonc-

tionner normalement. Dans cette logique, il n'est pas utile de donner la même importance à tous les composants. C'est par exemple le cas lorsqu'une architecture matérielle possède ses propres mécanismes de résilience : l'arrêt d'un des composants ne suppose pas nécessairement l'arrêt du service auquel il participe. Superviser devrait induire une vue analytique du système d'information ;

- les systèmes de supervision procurent également un moyen d'améliorer la qualité des services : c'est au travers des données qu'ils collectent que l'on peut identifier des points d'amélioration et en mesurer l'efficacité.

Si ITIL donne un certain nombre de préconisations qui permettent de déduire ce que pourrait être un système de supervision, il ne définit pas réellement les moyens à mettre en œuvre. La difficulté reste donc d'implémenter ces conseils de bonne pratique, tout en tenant compte des spécificités de l'organisation. Par exemple, la définition systématique des objectifs, entre autres celle d'un fonctionnement « normal », n'est pas clairement établie à Grand Besançon Habitat : concrètement, il n'existe que très peu de SLA et d'accords formels. Non que cela soit pénalisant dans les faits, mais il en ressort un certain flou quant à l'évaluation effective du fonctionnement du SI. La démarche entamée ici, dans le cadre de ce mémoire, garde donc une portée expérimentale, dans le sens où elle ne procède pas d'une application stricte de ces bonnes pratiques, mais vise plutôt à en évaluer les possibilités et les écueils. En partant de divers cas d'espèce, le service informatique a donc tenté de ré-assembler les divers composants du SI dans cette perspective de « service ». De fait, il s'agit d'amorcer une analyse du SI en tâchant, non pas de valoriser uniquement ses composants techniques, mais davantage de s'attacher à leurs fonctions.

Ces dernières sont représentées par un organigramme fonctionnel qui restitue de façon globale les rôles de chacun des salariés, répartis en services. C'est ce que l'on observe dans l'Annexe 6.1.

La logique du SI ne vise toutefois pas un découpage aussi strict, puisque pour faciliter le transfert d'informations, certaines ressources sont nécessairement communes. Plus précisément, les ressources du SI pourraient être représentées comme un emboîtement de composants interdépendants, ainsi que le montre l'Illustration 2.1.

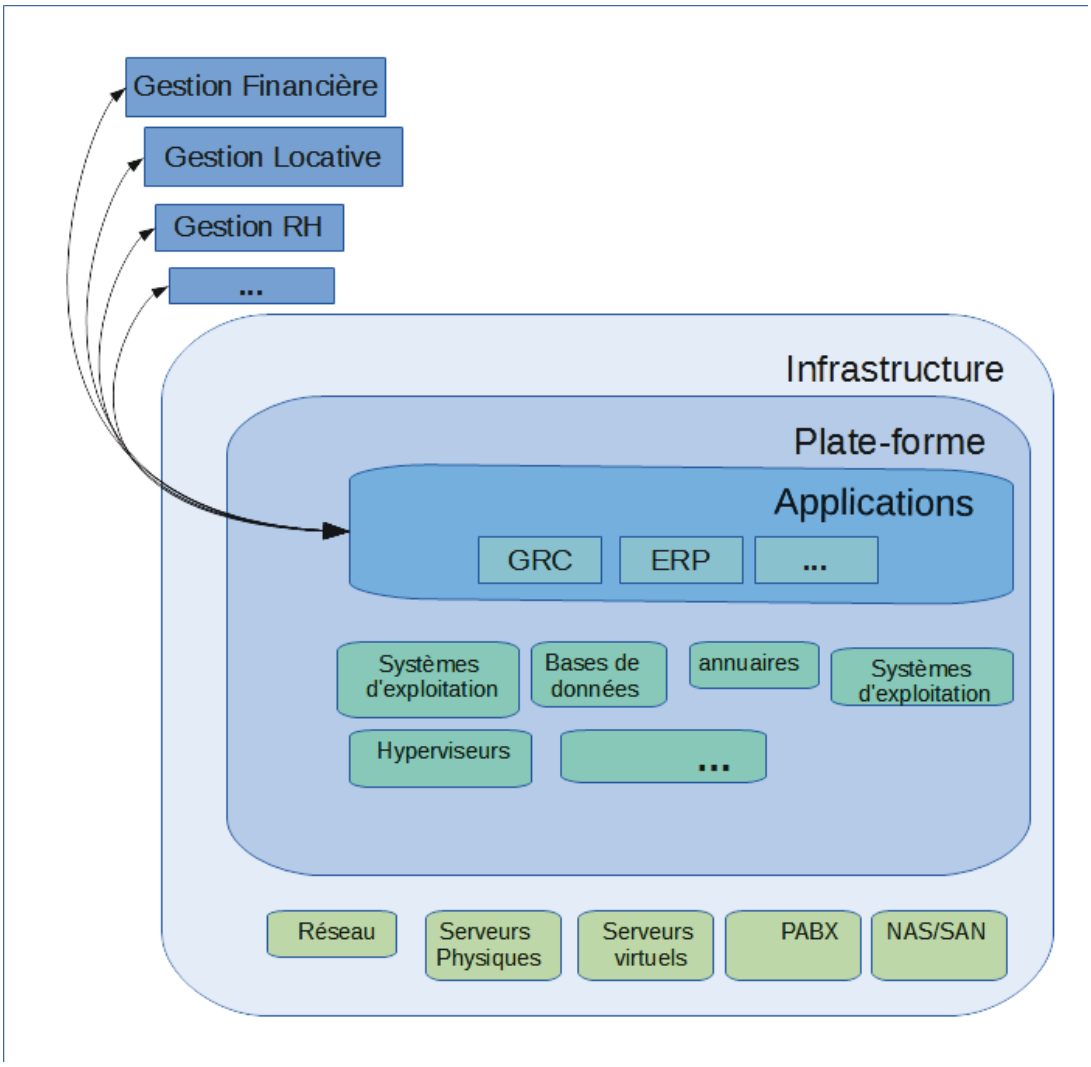


Illustration 2.1 : schéma des composants du SI

En pratique, il est effectivement difficile d'obtenir une vue exhaustive de tous les composants du SI permettant de les distinguer en termes d'importance ou de dépendances.

Par exemple, on ne peut dissocier une application du système d'exploitation sur lequel elle repose, ni même dissocier l'infrastructure des applications et des services qui y sont associés. Partant d'objectifs fonctionnels, l'analyse a permis de revenir, au fur et à mesure, à ce que pourrait être un socle commun.

2.2. Évaluation des besoins

L'intention du projet de supervision vise à identifier les principales cibles en s'appuyant sur une vision fonctionnelle du SI. L'identification des processus et mécanismes mis en œuvre pour délivrer un service emprunte à l'urbanisation des systèmes d'information les différents points de vue fonctionnels, applicatifs et techniques qui permettent de dissocier le SI en différentes strates[19]. Le point de vue métier, non qu'il soit de moindre importance, revêt des caractéristiques plus complexes qui ne concernent pas systématiquement l'infrastructure informatique. L'exemple qui suit constitue une sorte de cas pratique qui illustre le cheminement analytique préalable à la mise en place d'un système de supervision. Ce sont des évaluations de ce type qui ont permis de cerner les composants les plus sensibles en termes d'activité métier.

2.2.1. L'exemple de l'ERP *Aravis*

L'activité de gestion locative désigne au sens large tout ce qui a trait à la gestion des bâtiments et des locataires. Chez GBH, c'est un ERP spécifique, *Aravis,* qui prend en charge la quasi totalité de ces aspects. Cet exemple est restitué ici de façon à illustrer la démarche analytique qui a permis d'isoler les principaux éléments à superviser.

2.2.1.1. Présentation

ERP signifie *« entE.R.P.rise ressource planning »*, il est l'équivalent en français de PGI qui signifie « Progiciel de Gestion Intégré ». C'est un progiciel qui permet de gérer l'ensemble des processus opérationnels d'une entreprise, en intégrant l'ensemble de ses activités (activités dites verticales telles que la production et l'approvisionnement ou bien horizontales comme le marketing, les forces de vente, la gestion des ressources humaines, etc.) autour d'un même système d'information.

Le terme ERP est apparu en 1970. Il est issu d'une méthode qu'il exploite, la méthode MRP utilisée dans la gestion et la planification de la production industrielle. Les progiciels de gestion intégrés proposent généralement des outils de *groupware*[10] et de *workflow*[11] afin d'assurer la transversalité et la circulation de l'information entre les différents services de l'entreprise.

10 Ensemble d'outils destinés à favoriser le travail en équipe. Comme par exemple des logiciels permettant à un groupe d'utilisateurs de travailler en collaboration sur un même projet sans être nécessairement réunis.

11 Un *workflow* est un flux d'informations au sein d'une organisation, comme par exemple la transmission automatique de documents entre des personnes.

Aravis est un ERP développé par un GIE dénommé ACG[12]. *Aravis* a été conçu sous forme modulaire et prend en charge la gestion financière et comptable, la gestion immobilière, la promotion du patrimoine et l'entretien (états des lieux, réparations, prise en charge des communs, etc.). *Aravis* n'est pas la seule application du parc de GBH, mais elle propose un certain nombre d'interfaces pour communiquer avec d'autres partenaires, par exemple en favorisant l'intégration et l'export de données ou en proposant des connecteurs. L'ensemble nécessite qu'une infrastructure spécifique soit disponible pour assurer son fonctionnement.

Initialement développées sous OS/400, les interfaces utilisateurs *Aravis* ont ensuite été développées en C++ de façon à fournir une IHM pour les utilisateurs de systèmes Windows. Une partie de l'application s'exécute donc sous des systèmes Windows, et une autre sur un *IBM Power i7* OS/400.

Aravis fournit donc différents types d'interfaces : la première est dédiée aux états des lieux et peut fonctionner en mode déconnecté, la seconde est une interface web portée sur la gestion de relation client (GRC). L'interface principale, quant à elle, est une IHM disponible pour n'importe quel client connecté au réseau, doté des droits suffisants et de l'application cliente.

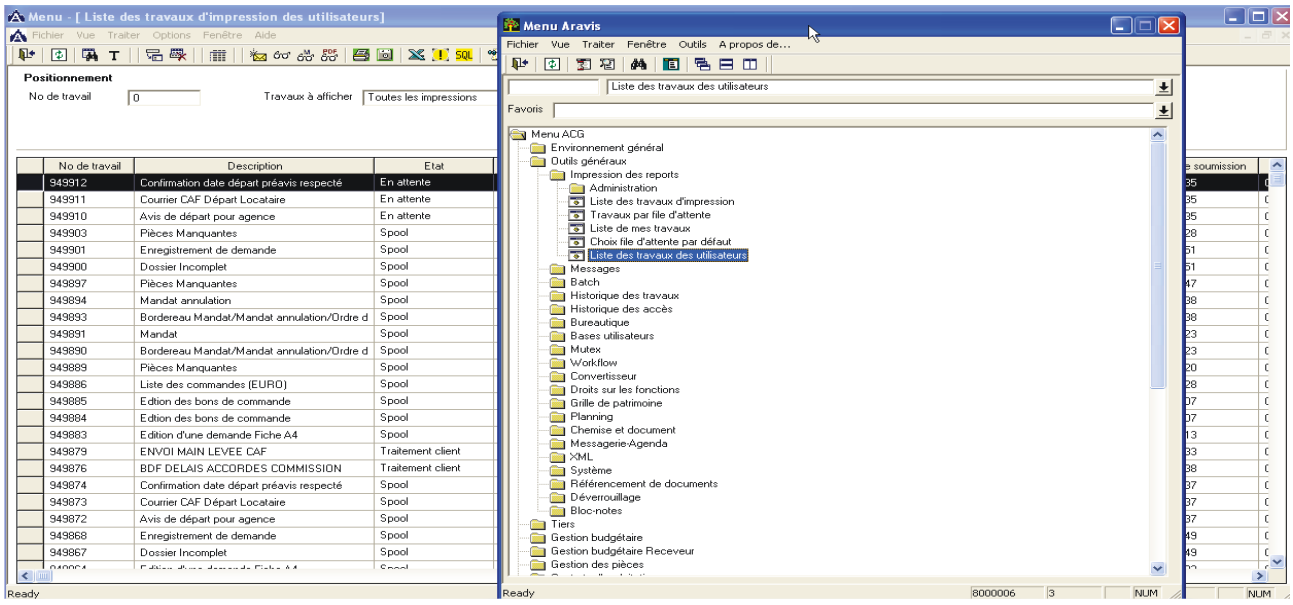


Illustration 2.2 : exemple de vue de l'IHM Aravis

12 ACG-synergie : http://www.acg-synergies.fr/

Aravis est liée à des applications tierces, comme :

- les outils bureautiques *(MS Office, LibreOffice ou Crystal Report)* et des modules de génération de documents : publipostage, éditions de listes, statistiques de production ;
- *use it flow* pour la gestion des transferts avec les partenaires externes : Centre des finances publiques, MSA, CAF, serveur de la demande unique ;
- une série de partages SMB[13] Windows qui centralisent des DLL[14] nécessaires au programme lui-même, ou des données spécifiques comme des plans de logements, des diagnostics de performance énergétique ou des autorisations de prélèvement SEPA ;
- une chaîne décisionnelle qui comprend un *datawarehouse* et des *datamarts* ainsi que les outils afférents, notamment un ETL et divers outils de *reporting* ;
- des liens plus diffus avec des applications comptables spécialisées dans la gestion du financement du patrimoine *(Sage)* ;
- un moteur d'impression, chargé d'extraire les données métier des base et de les assembler selon des modèles prédéfinis en vue de produire différents formats ou d'imprimer ;

Un examen rapide de la couverture applicative de GBH dont le schéma figure en Annexe 6.2 permet de mieux appréhender l'étendue fonctionnelle des différents modules de l'application *Aravis* et son impact sur l'activité métier de GBH.

13 SMB ou S*erver Message Bloc* est un protocole réseau de partage de fichiers.

14 DLL : D*ynamic Linked Library*. En raison de fréquentes mises à jour, ces DLL *Aravis* sont centralisées. *Aravis* ne fonctionne donc qu'au moyen de ces partages.

2.2.1.2. Aspects fonctionnels et applicatifs

On note que nombre des fonctionnalités de l'ERP *Aravis* reposent sur la présence de soutènements architecturaux : serveur SMTP, connectivité internet, services d'impression centralisés, etc. De ce fait, l'usage d'*Aravis* constitue non seulement un intérêt notable du point de vue fonctionnel et métier, mais également du point de vue de l'infrastructure technique qu'il sollicite, directement ou non. L'Illustration 2.3 schématise cette distinction entre les fonctions métier et les modalités d'accès à la couche dite « applicative ».

Illustration 2.3 : représentation schématique des fonctions Aravis

La seconde vue applicative schématisée dans l'Illustration 2.4 permet de dégager des logiques d'arrière-plan, c'est-à-dire celles qui ne sont pas visibles par les utilisateurs. Parmi elles, on dénombre notamment l'outillage décisionnel, *datawarehouse* et *datamart*, les outils d'automatisation d'échange de données avec les partenaires externes, ou encore les instances applicatives liées à la gestion des mails - serveurs IMAP, SMTP, systèmes antivirus et *firewall*.

À ce point, il apparaît non seulement évident que la notion de système et d'interdé-

pendance est pertinente, mais également que superviser une instance applicative aussi diffuse dénote une certaine complexité. En effet, en élongeant les liens de dépendance applicatifs jusqu'à la couche système, il n'apparaît que rarement un lien unique entre une « ressource » applicative et l'instance système sur laquelle elle repose. C'est par exemple le cas des bases de données dont la *fonction technique* peut être mise en œuvre pour plusieurs *fonctions métier distinctes* de façon simultanée.

Le premier problème évident tient à la complexité des dépendances entre les couches applicatives et techniques. D'un côté le SI dispose de ses propres objectifs fonctionnels (sécurité, gestion des données, transferts, etc.) et de l'autre, ceux-ci doivent être le support des activités métier. L'ambiguïté de ces distinctions oblige à se focaliser sur des fonctions globales, de façon à ne pas induire de confusions entre les fonctions du SI et celles de l'organisation. Typiquement, les logiques de *cluster* trouvent difficilement leur place dans ce type de schéma, ou nuisent à sa lisibilité. De même, la notion de sécurité ne revêtirait pas le même sens selon qu'elle concerne le SI ou l'activité de bailleur social.

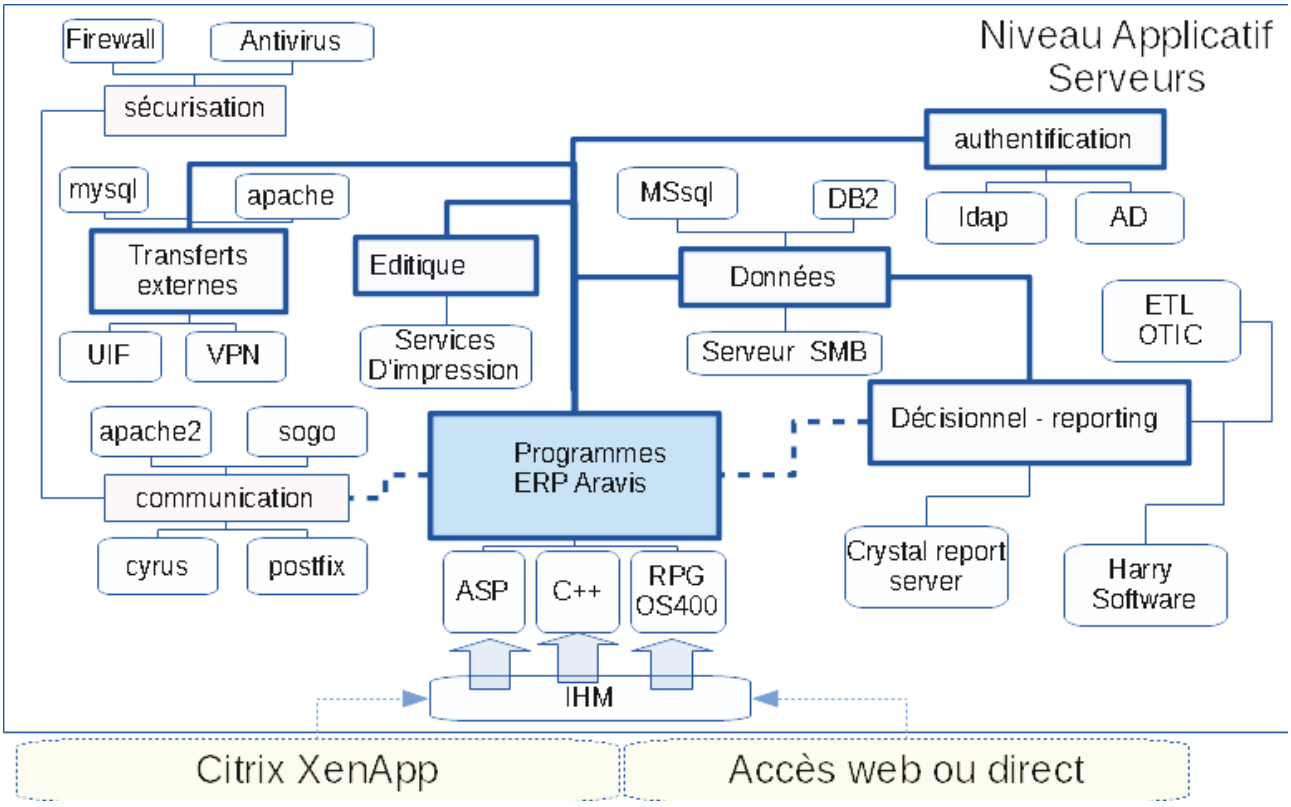


Illustration 2.4 : schéma simplifié des dépendances applicatives de l'application Aravis

Au-delà de ces distinctions sémantiques, la représentation conceptuelle sur laquelle nous nous appuyons vise à montrer que chaque groupe d'applications possède ses propres fonctions : transferts externes, éditique, *reporting*, archivage, etc. Ces dernières

sont organisées dans le but de mettre à disposition des fonctions exploitables par les processus métier. À l'inverse, cette logique applicative repose elle-même sur une infrastructure spécifique. C'est à ce point de vue que nous nous attachons maintenant.

2.2.1.3. Vue technique : plates-formes, systèmes et infrastructure

On désigne le socle « technique » comme l'ensemble des composants nécessaires à la mise en œuvre de la couche applicative. Cela concerne donc les composants matériels, les systèmes d'exploitation ainsi qu'une couche intermédiaire constituée des logiques applicatives d'arrière-plan.

Cette couche est désignée dans l'Illustration 2.1 de façon générique comme celle des « plates-formes ». On y retrouve un découpage logique qui distingue les différents éléments techniques du SI selon leur nature. De façon synthétique, on distingue :

- les « serveurs » : il s'agit des instances applicatives qui implémentent différents services comme les serveurs web ou les bases de données. À la différence du point de vue applicatif, qui valorise des instanciations spécifiques de serveurs, on considère ici les serveurs d'un pur point de vue technique. Typiquement, le rôle d'un serveur Microsoft IIS revient à répondre à des requêtes HTTP. Du point de vue applicatif, il revient à mettre en œuvre un outil spécifique de gestion client qui est une instance IIS, une implémentation Microsoft de serveur web. Du point de vue métier, l'objectif est le suivi des relations avec les locataires ;
- les systèmes d'exploitation qui reposent de façon quasi systématique sur une couche de virtualisation. Dans le parc de GBH, on dénombre trois instances *VMWare ESXi* et deux instances *Xenserver*[15]. Ces instances fonctionnent en *cluster*[16] ; par exemple, chaque machine virtuelle peut être déplacée sur l'un ou l'autre des membres du *cluster* selon les besoins. Cette capacité de résilience est rendue possible par la couche de virtualisation, mais également par les redondances matérielles mises en œuvre sur l'*IBM BladeCenter H*[17] ;
- le réseau constitue également une structure fondamentale : il assure la communication entre les serveurs et les points d'accès délocalisés. L'Illustration 2.5 schématise les composants en marquant les accès disques par fibre optique et les liens réseaux qui permettent la communication entre les différents sous-réseaux. Les services téléphoniques constituent un composant spécifique qui dépend de la couche réseau et d'un IPBX ;

15 *Xenserver* est un hyperviseur, au même titre que l'*ESX VMWare* : voir [20] , [21]

16 On fait référence ici au fonctionnement en grappe d'un groupe de serveurs qui sollicitent les mêmes matériels.

17 L'*IBM BladeCenter H* est un châssis qui peut intégrer 12 serveurs lames et plusieurs modules redondants (fibre optique, réseau, alimentation électrique) [22]

- les logiques de stockage qui sont mutualisées dans des NAS ou des SAN.

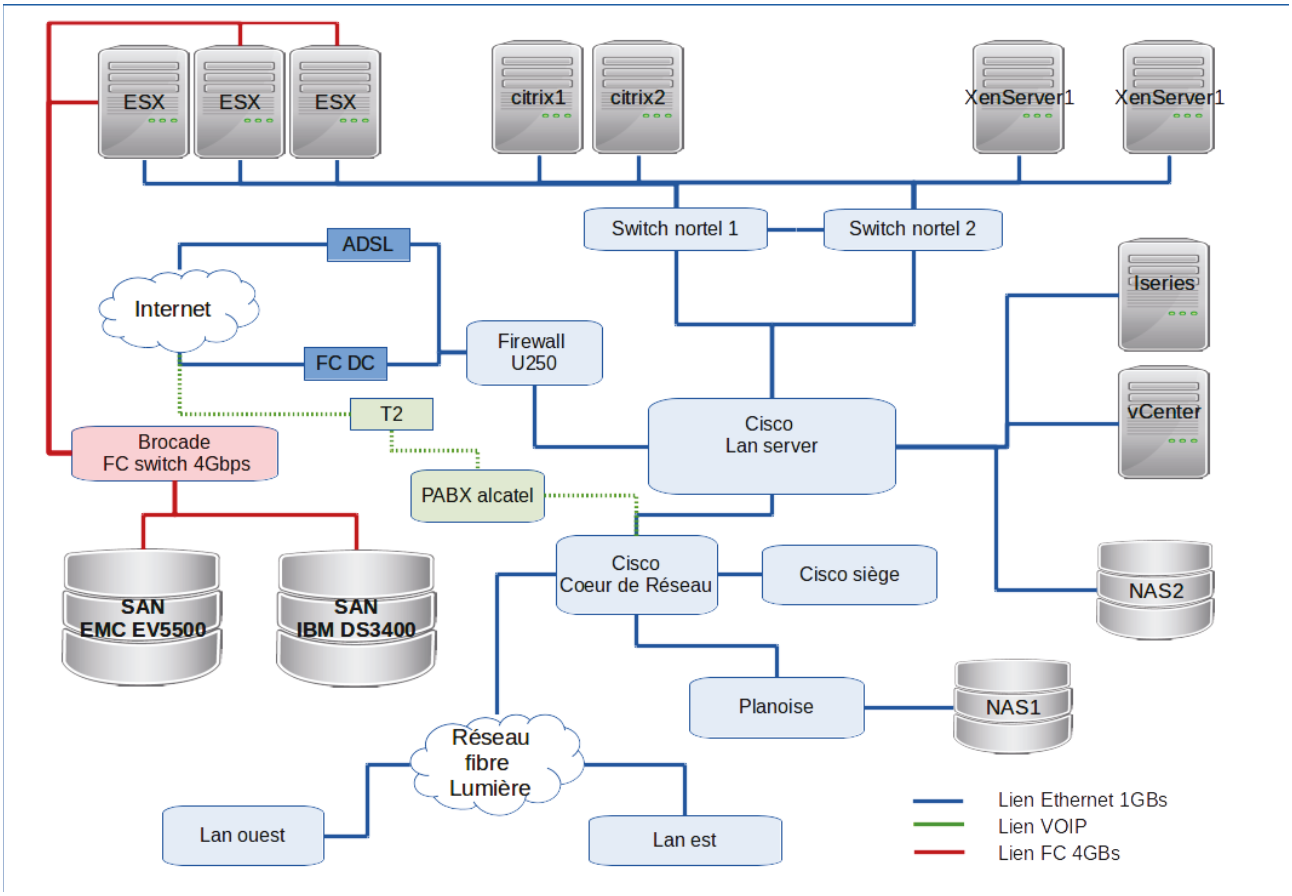


Illustration 2.5 : schéma simplifié de l'architecture technique

Ces éléments sont repris dans des cartographies élaborées par le service informatique de GBH et illustrent les différents réseaux. Toutefois, cette façon de faire pose la question de la pertinence de la représentation. Si certaines parties font l'objet d'une cartographie détaillée (voir les annexes 6.2, 6.3, 6.4 et 6.5), leur entretien reste fastidieux tant l'infrastructure peut être amenée à évoluer rapidement. Au final, le risque de disposer d'une cartographie obsolète demeure important. Cet état de fait renvoie l'idée que le SI est en constant mouvement, en changement continu. Aussi, le figer dans une représentation trop spécifique reviendrait à réfuter cette caractéristique fondamentale. Les cartographies ont toutefois l'avantage de laisser apparaître différentes vues : réseau, zone spécifique (test, DMZ, production), applications, machines virtuelles, stockage, etc. Bien qu'elles soient moins adaptées aux questions de détail, elles peuvent mettre en évidence les aspects les plus sensibles du SI.

En l'occurrence, le stockage mutualisé sur les SAN fibre optique, le réseau interne et la couche de virtualisation constituent des ensembles disposant de leurs propres méca-

nismes de résilience : liens redondants, disques RAID, *cluster* logique d'hyperviseur, etc. Par rapport à une application, la question se mesure davantage en termes d'*impact* qu'en termes de *liens de dépendance* : par exemple, la défaillance de l'ensemble du réseau induit la défaillance de la quasi-totalité des services du SI.

Nous constatons en outre que le socle technique, à l'instar de l'activité métier, dispose en fait de ses propres fonctions. Ces fonctions sont décrites par les processus ITIL : capacité, disponibilité, continuité, niveau de service, etc. La cohérence de ce « socle » tient à celle de ses composants : réseaux, virtualisation, systèmes et serveurs. En conséquence, les logiques d'intégration mises en œuvre, comme par exemple les *clusters* d'hyperviseurs *ESXi* ou la mutualisation du stockage dans les SAN, ne permettent pas de valoriser une relation simple entre une application et les composants de la plate-forme technique.

2.2.2. Synthèse

En se limitant à une vision globale du système, il apparaît qu'une application complexe comme l'ERP *Aravis* tend à solliciter un grand nombre de composants logiques et de ressources physiques, du fait de l'étendue de sa couverture fonctionnelle. Du point de vue de l'infrastructure informatique, plusieurs mécaniques sont en jeu : redondances de liens réseau, répartition de la charge et déplacement des machines virtuelles sur le *cluster ESX*i, etc. Il s'agit d'un point de vue très large, qui tend à englober le fonctionnement de l'application comme une conséquence de la mise en œuvre de l'infrastructure.

utilisateurs	Application	Composants logiques	**systèmes**	**infrastructure**
Siège, tous services	*Aravis* *Aravis - GRC*	• Partages SMB • IIS • services mail • Annuaires • Transferts UIF • BDD Mysql, pgsql	• Windows 2008R2 • CentOS 6 • debian Linux 6 et 7 • IBM OS/400	• virtualisation *ESX* • virtualisation Xenserver • vCenter • LAN agences • LAN CFP • Stockage SAN • accès internet externe
Agence et CFP		• Citrix XenApp • Partages SMB • IIS • services mail • Annuaires		
Siège, service direction, finance, comptabilité, patrimoine	reporting et analyse : Harry, crystal Report	• Chaîne décisionnelle : ETL • BDD : DB2 MSSQL		

Tableau 2.1 : composants assurant les fonctionnalités déployées par l'ERP Aravis

De façon assez schématique, dans le Tableau 2.1, un premier jeu de dépendances émerge entre les strates fonctionnelles et applicatives. Au niveau de l'infrastructure, si l'on peut isoler quelques dépendances évidentes, il paraît plus clair d'envisager une approche en termes d'impact, tant les composants sont intégrés.

L'objectif étant de cibler les principaux enjeux, nous chercherons à évaluer et la question de l'impact et celles des liens de dépendance logique avec l'outil de supervision. À ce stade du projet, les ressources techniques suivantes ont été inventoriées comme des ensembles cohérents, ayant un impact critique sur l'ensemble du SI :

- **les liaisons réseau**, qui impliquent tous les commutateurs, les éléments actifs spécifiques comme le firewall Netasq U250, les différents LAN[18] et la connectivité internet qui repose sur des modems (SDSL, ADSL ou fibre dédiée). La quasi-totalité des applications des postes de travail s'appuient sur une logique client-serveur n-tiers, conférant aux services réseau un caractère critique ;
- **les couches de virtualisation** *ESXi[23]* (zone de production) ou *XenServer[24]* (DMZ[19] et test) qui maintiennent une soixantaine de serveurs ;
- **la couche système** qui concerne plus directement les systèmes d'exploitation, principalement des systèmes Linux *CentOS* ou *Debian* et Windows 2008R2. Un Iseries Power 7 est également utilisé en production. On recense, mais de façon plus anecdotique des systèmes IOS (Cisco)[26] ou Netbsd[27];
- **la couche plate-forme** qui concerne les éléments applicatifs en arrière-plan : serveurs web *(NGiNX, Apache, IIS)*, bases de données *(MSSQL, MySQL, PostgreSQL)*, serveurs mail *(Postfix, Cyrus)*, etc. ;
- **la couche applicative** qui concerne les applications en avant-plan, le plus souvent des interfaces utilisateurs ou des instances applicatives spécifiques. On y trouve par exemple les clients de messagerie, les interfaces graphiques des clients de l'ERP *Aravis*, ou les navigateurs, qui permettent d'accéder à diverses interfaces web.

On peut ensuite lier à la couche applicative un certain nombre de macro-fonctionnalités qui se rapportent directement à l'activité de l'entreprise : on se situe donc plus directement dans la vision fonctionnelle. Dans notre cas, les fonctions suivantes ont été mises en évidence :

18 LAN : *Local Area Network*, se réfère aux différents sous-réseaux constituant le réseau interne.

19 DMZ : zone démilitarisée (de l'anglais *DeMilitarized Zone*) est un sous-réseau séparé du réseau local et isolé de celui-ci et d'Internet par un pare-feu. Ce sous-réseau contient les machines auxquelles internet est susceptible d'accéder.[25]

- gestion locative : repose très directement sur l'ERP *Aravis ;*
- échanges internes : stockage, diffusion et échanges de données, archivage ;
- logiques financières et comptables ;
- logiques de transfert et d'échange : accès externes et échanges de données avec des partenaires externes, services externalisés ;
- décisionnel : exploitation des données et production de statistiques ;
- gestion des moyens généraux : ressources humaines et activités de supports transverses.

Ces fonctions gardent une connotation générique qui permet d'établir un lien de façon schématique avec la couche applicative, ainsi que le montre l'Illustration 2.6.

Les items représentés dans cette Illustration 2.6 sont une abstraction qui facilite le passage à la vision métier, toujours dans l'idée d'assurer un alignement stratégique. En deçà des visions métier et applicatives, émergent les aspects plus « techniques » du SI : systèmes, matériels et plates-formes intermédiaires.

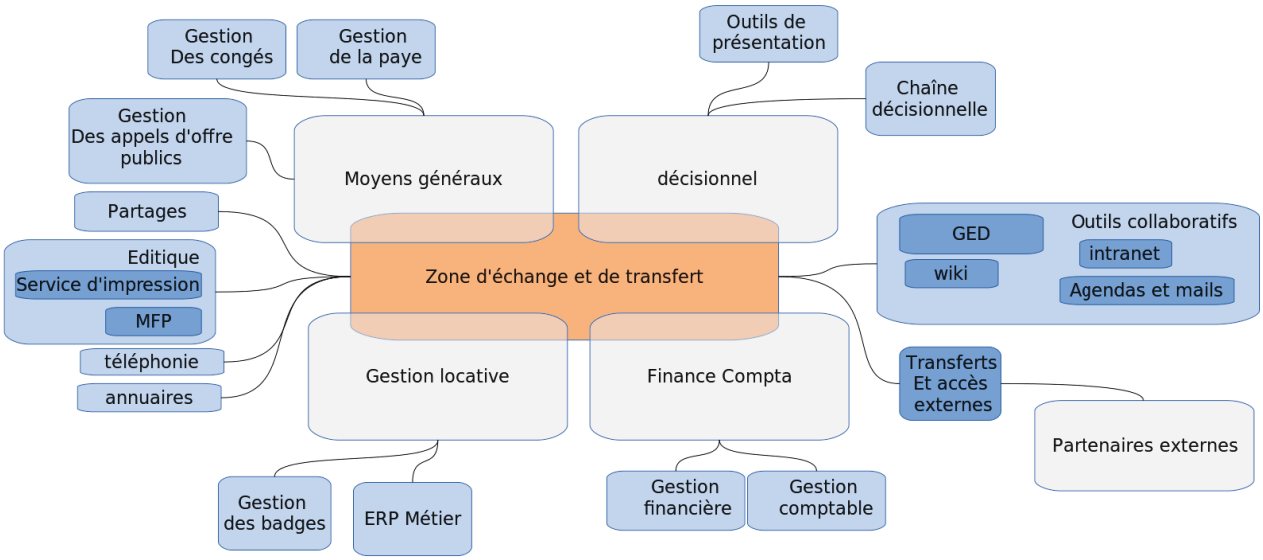


Illustration 2.6 : vision fonctionnelle

Si la partie matérielle est globalement dotée de certaines facultés de résilience, quelques composants restent fragiles, soit en raison de la faible fiabilité du matériel (par exemple matériel vieillissant dont le taux de pannes augmente), soit en raison d'instabilités notoires, comme c'est le cas pour certains logiciels dont le développement a été abandonné. Quoiqu'il en soit, ces déficiences sont clairement identifiées et restent rares.

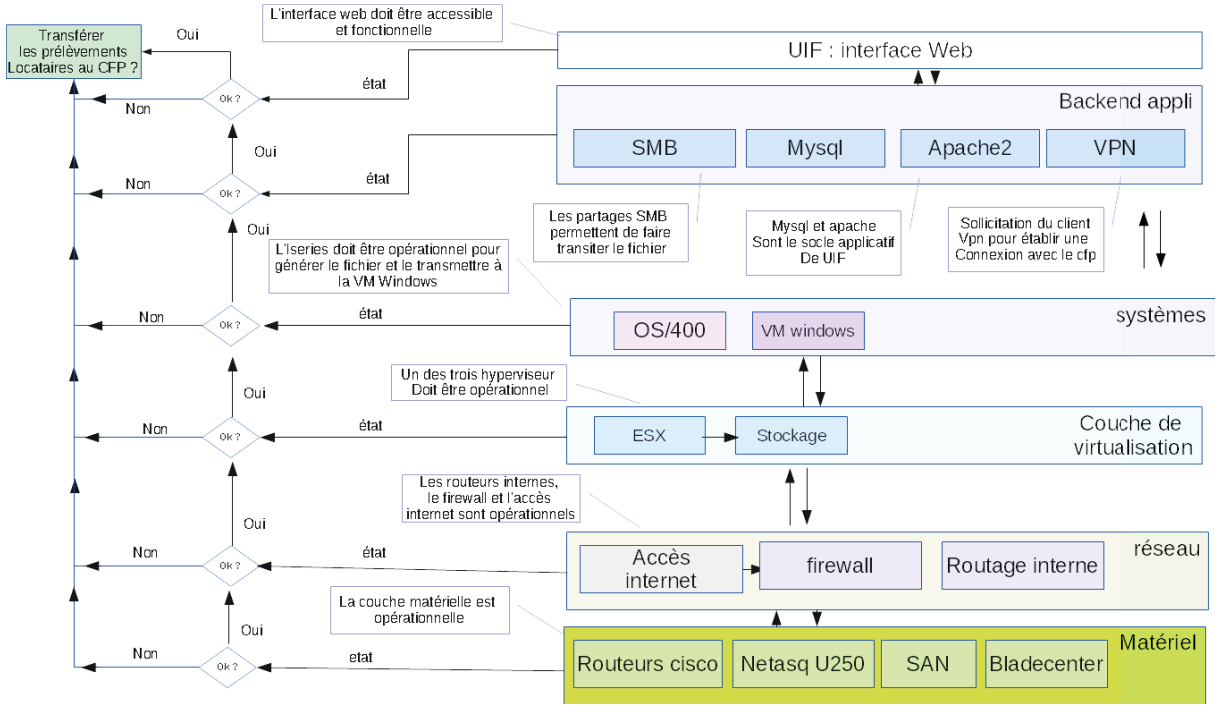


Illustration 2.7 : exemple d'analyse de dépendances : UIF, application de transferts sécurisés

Dans le cas présent, en référence aux définitions ITIL, les difficultés consistent en :

- vérifier l'état de fonctionnement de composants relativement hétérogènes : systèmes de virtualisation, OS/400, Windows, Linux, BSD ;
- mesurer les performances des différents composants de façon homogène pour en faciliter la comparaison et la constitution d'éventuels KPI ;
- établir des seuils d'alerte, c'est-à-dire déterminer les symptômes d'un événement normal, inhabituel ou d'une exception ;
- les regrouper de façon cohérente en s'appuyant sur la logique de *services* semblable à celle que l'on retrouve dans les préconisations ITIL ;
- prioriser ces services ou groupes logiques, notamment en évaluant l'impact de leur fonctionnement sur l'activité du SI de Grand Besançon Habitat ;
- évincer les dépendances redondantes : il s'agit d'isoler artificiellement une représentation de ce qu'est un composant ou un service. Concrètement, un service quel qu'il soit ne peut fonctionner sans la couche de virtualisation ou le réseau.

L'évaluation d'un service peut donc reposer sur une série d'assertions dont le chaînage logique représenterait ces interdépendances. D'évidence, ces évaluations nécessitent une logique homogène. C'est à ce niveau que l'on peut s'appuyer sur la typologie des

événements ITIL *(normal, inhabituel ou anormal)* qui fournit un niveau d'abstraction suffisamment important.

L'Illustration 2.7 montre comment les logiques d'interdépendances peuvent être schématisées pour un « service » que l'on nomme « transfert des prélèvements au CFP». Ce service consiste à automatiser le transfert de données décrivant les ordres de prélèvement des comptes bancaires des locataires au centre de finances publiques. Du point de vue fonctionnel, on associe ce service aux logiques de transferts externes. L'idée est de pouvoir évaluer à quelles conditions l'application en charge d'effectuer les transferts automatiques de données vers le VPN du centre des finances publiques est opérationnelle. On distingue les différentes couches décrites plus haut et, pour chacune d'entre elles, les composants techniques sollicités. Dans ce cas précis, on suppose que l'état du service peut être évalué par un KPI qui restitue un état global en fonction de l'état de différents composants.

Plus en détail, si l'on s'attache au composant *routeur*, on pourrait considérer que ce dernier est opérationnel en effectuant les opérations logiques suivantes :

```
SI
      (
              (le routeur répond à une requête ICMP)
              ET ( le routeur peut renvoyer l'état de certains de ses ports)
              ET ( que ses capacités techniques internes ( mémoire, disque, processeur) présentent
                      des seuils de fonctionnement acceptables)
              ET ( que ces ports ne présentent pas d'erreur notable)
      )
ALORS
      (l'état du routeur est normal)
SINON
      (L'état du routeur est anormal ou inhabituel)
)
```

Dans la même idée, en utilisant le retour d'état sur chacun des composants, on peut le propager au service et en déduire l'état du service « transférer un fichier de prélèvement au CFP ». Cette approche, bien qu'apparemment fastidieuse, laisse apparaître la nécessité de factoriser les tests de supervision. Conjointement, cela explique que l'on isole chacun des composants dont le rôle est identifié comme critique pour l'activité de production.

Le schéma n'entre pas dans le détail des informations qui permettent de déduire l'état des composants et de le comparer à des seuils déterminés - c'est un point que nous détaillerons plus loin et qui répond à diverses questions relatives à la gestion des capacités. On observera également comment récupérer des données de performance indispensables aux processus de gestion des capacités ou à l'évaluation de la qualité du service.

En fin de compte, ces analyses ont permis de :

- identifier les ressources et composants fondamentaux : le réseau et la couche de

virtualisation, les systèmes, les plates-formes et les applications ;

- dégager des groupes applicatifs cohérents et dépendants d'une série structurée de composants techniques ;
- dégager une logique de décomposition des différents éléments du SI, en notant l'importance des liens de dépendance qu'ils entretiennent ;
- poser l'idée qu'un service peut être évalué par une expression logique regroupant ces indicateurs ;
- questionner la lisibilité et la sémantique des indicateurs. Le SI recèle une certaine complexité, qu'il s'agisse de dépendances matérielles ou logiques : établir un lien entre les composants de l'infrastructure et la vision métier induit une abstraction. C'est à ce niveau que se pose la question du sens : chacun des acteurs du SI, utilisateur ou administrateur, construit des représentations qui lui sont propres. Il n'est donc pas envisageable de restreindre le SI à une seule représentation.

Ces difficultés, qui s'expriment en termes d'analyse, de conceptualisation ou de représentation, posent clairement la question du choix de l'outil de supervision : si proposer des mesures objectives de performances des composants techniques semble évident, l'évaluation de l'alignement stratégique reste plus délicate, notamment parce qu'elle suppose d'abstraire le SI de sa réalité technique et d'intégrer un *processus itératif d'adaptation[28]* dans l'outil même. En ce sens, il a semblé approprié de mettre en œuvre une solution de supervision capable d'offrir des facilités d'évolution et d'adaptation.

2.3. Choix d'une solution de supervision

Si l'analyse permet de relever un certain nombre de difficultés, elle met en exergue la nécessité d'appréhender la supervision comme un processus continu et évolutif. Outre cette souplesse, l'outillage de supervision doit pouvoir restituer une mesure comparable d'éléments hétérogènes et en fournir une représentation synthétique.

Dans cette idée, il est important de pouvoir s'appuyer sur des protocoles supportés par l'ensemble des composants. S'agissant ici d'un réseau local TCP/IP, on s'intéressera plus spécifiquement aux protocoles compatibles avec ce type de réseau, par ailleurs fort répandu. Cette couche réseau, ainsi que nous l'avons noté précédemment, constitue un dénominateur commun pour l'ensemble des composants de l'architecture de Grand Besançon Habitat. Un premier examen de quelques-uns de ces protocoles, comme SNMP, est indispensable pour évaluer la pertinence des solutions de supervision au regard des

spécificités de l'architecture de GBH. En second lieu, nous observerons, au regard de ces éléments, comment s'est effectué le choix d'une solution de supervision.

2.3.1. Protocoles et implémentations utilisés dans les procédés de supervision

2.3.1.1. Le cas du protocole SNMP

SNMP est l'acronyme de *Simple Network Management Protocol*. C'est un protocole réseau qui a été défini par un groupe de travail de l'IETF *(Internet Engineering Task Force)* dans le cadre de la définition d'un système de gestion pour les réseaux utilisant les protocoles TCP/IP. Ce protocole est extrêmement répandu aujourd'hui, spécialement pour le contrôle des réseaux locaux. On peut, de fait, le considérer comme un standard incontournable[29]. Trois versions de SNMP se sont succédé : SNMPv1, SNMPv2 et SNMPv3.

L'environnement SNMP est un protocole destiné à surveiller la performance du réseau, à détecter et à analyser ses fautes et à configurer ses éléments.

SNMP a été approuvé par l'IAB *(Internet Activities Board*), responsable des spécifications de TCP/IP. Ce standard est défini par une série de RFC, parmi lesquelles :

- RFC1155SMI : *Structure of Management Information ;*
- RFC 1156MIB : *Management Information Base ;*
- RFC1157 : *SNMP Protocol ;*
- RFC 2258 : *MIB II.*

Ce cadre définit trois composants fondamentaux :

- le protocole lui-même, situé au niveau application de l'architecture en couches de TCP/IP qui définit la structure formelle des communications ;
- la base d'information, ou MIB, qui recense l'ensemble des variables relatives aux matériels et logiciels supportés par le réseau et qui définit les objets de gestion dans l'environnement TCP/IP ;
- les spécifications de l'infrastructure de gestion, SMI *(Structure of Management Information)* qui définissent les modalités de représentation des ressources dans la MIB et la façon de les obtenir.

L'environnement SNMP est constitué de deux éléments principaux :

- une station de gestion, appelée NMS *(Network Management Station)* qui contient une base maître qui représente toutes les ressources du réseau et les informations

de gestion associées ;

- des agents logiciels qui sont en œuvre dans chaque station gérée. Ces agents peuvent représenter un serveur, un routeur, des équipements de transmission, etc. Chacun de ces agents est doté d'une MIB comprenant une base d'objets gérés et des variables.

Le protocole

Dans son fonctionnement, le protocole SNMP est asymétrique. Il est constitué d'un ensemble de requêtes, de réponses et d'un nombre limité d'alertes. Le manager envoie des requêtes à l'agent, lequel retourne des réponses. Lorsqu'un événement anormal survient sur l'élément réseau, l'agent envoie une alerte *(trap)* au manager.

SNMP utilise le protocole UDP [RFC 768]. Le port 161 est utilisé par l'agent pour recevoir les requêtes de la station de gestion. Le port 162 est réservé à la station de gestion pour recevoir les alertes des agents.

On dénombre trois types de requêtes : GET, SET et TRAP.

Les commandes de type GET REQUEST permettent aux NMS d'interroger des objets et des variables gérés dans la MIB des agents. Plus précisément, on distingue les commandes suivantes :

❒ la requête *GetRequest* permet la recherche d'une variable sur un agent ;

❒ la requête *GetNextRequest* permet aux stations de recevoir le contenu de l'instance qui suit l'objet nommé. Grâce à cette commande, la station de gestion peut balayer les tables de MIB ;

❒ la requête *GetBulk* permet la recherche d'un ensemble de variables regroupées. Cette requête a été introduite dans la version 2 du protocole SNMP, elle a pour but de limiter les flots d'informations ;

❒ la commande de type *GetResponse* est le message retourné aux NMS par les agents qu'ils interrogent, en réponse aux messages de type GET REQUEST, GET NEXT REQUEST et SET REQUEST .

La requête SET *(SetRequest)* permet de changer la valeur d'une variable sur un agent. Par exemple, cette commande autorise un manager à modifier la table de routage d'un hôte.

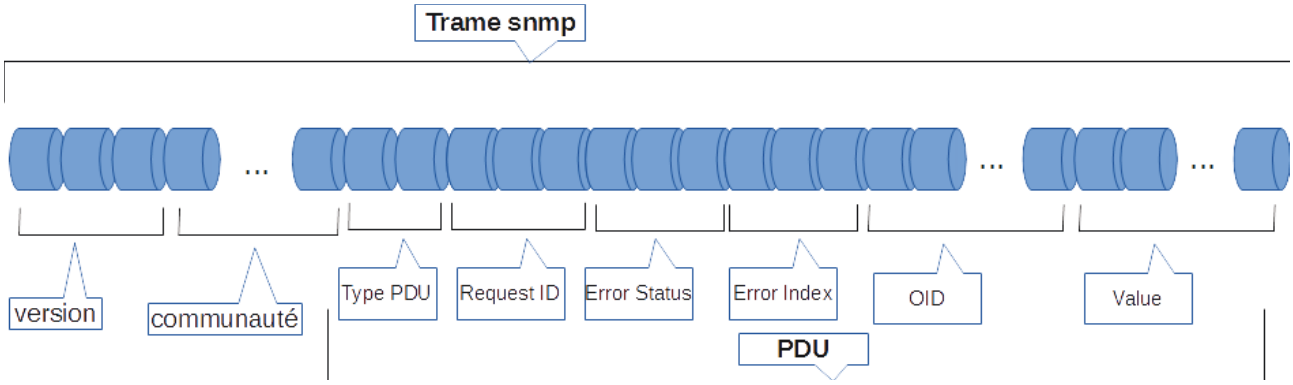


Illustration 2.8 : structure d'une trame SNMPv1

La trame SNMPv1 est complètement encodée en ASN.1 et sa longueur est variable. Les requêtes et les réponses ont le même format. La communauté permet de définir des domaines d'administration. Elle est constituée d'une chaîne de caractères dont la valeur par défaut est « public ». Dans le PDU SNMP (*Protocol Data Unit*) décrit dans l'Illustration 2.8, la zone « type PDU » correspond au type de requête décrit plus haut. On retrouve ainsi la nomenclature de l'Illustration 2.9. Le champ « Request ID » permet à la station de gestion (NMS) d'associer les réponses à ses requêtes. Les champs *Error* permettent de prendre en compte une erreur renvoyée par l'agent : *NoAccess, WrongValue, WrongLength*, etc.).

L'OID quant à lui se réfère plus directement à un nœud de la MIB évoquée plus haut.

Type de PDU	Nom
0	Get-request
1	Get next-request
2	Set-request
3	Get response
4	Trap

Illustration 2.9 : correspondance type de PDU - type de requête SNMP

La version 2 essaye principalement de limiter le flot d'informations induit par les requêtes SNMP en introduisant une requête GETBULK et une requête GET améliorée. C'est également la version 2 qui introduit la possibilité de communiquer entre différents managers et la commande de type INFORM *(trap)* qui permet d'envoyer des alertes quand un événement non attendu se produit sur l'agent. À son initiative, celui-ci envoie

une commande *trap* à la station NMS. Concrètement, il s'agit d'informer le NMS d'événements de type « arrêt de l'agent, dépassement d'un seuil, remontée d'un lien réseau, etc. ».

En 1996, l'IETF a formé un nouveau comité en vue d'examiner les problèmes de sécurité dans SNMP. Le standard SNMPv3 voit le jour en 1998. Cette version reprend les améliorations de SNMPv2 en incluant de nouveaux éléments de sécurité, ces derniers constituant de fait la principale amélioration. SNMPv3 est composé de trois modules :

- *Message Processing and Control*, qui définit la création et les fonctions d'analyse des messages ;
- *Local Processing,* qui s'occupe de la question des contrôles d'accès et de l'exécution des données ;
- *Security*, qui permet l'authentification et le chiffrement des messages.

En pratique, tous les agents ne supportent pas nécessairement toutes les commandes, ni même toutes les versions de SNMP. Selon les agents, ou le type de matériel utilisé, il reste difficile de ne se reposer que sur la version 3 de SNMP, les versions 1 et 2 étant encore largement utilisées.

Les MIB

La MIB *(Management Information Base)* est la base de données des informations de gestion maintenue par l'agent, auprès de laquelle le NMS va récupérer des informations.

Un fichier MIB est un document texte écrit en langage ASN.1[30] [31] *(Abstract Syntax Notation 1)* qui décrit les variables, les tables et les alarmes gérées au sein d'une MIB. Par exemple, une interface réseau se décrit de la façon suivante dans une base MIB :

```
ifDescr OBJECT-TYPE
            SYNTAX  DisplayString (SIZE (0..255))
            ACCESS  read-only
            STATUS  mandatory
            DESCRIPTION
                    "A textual string containing information about the
                    interface.  This string should include the name of
                    the manufacturer, the product name and the version
                    of the hardware interface."
            ::= { ifEntry 2 }
```

La MIB est une structure arborescente dont chaque nœud est défini par un nombre ou OID *(Object IDentifier)*. Elle contient une partie commune à tous les agents SNMP en général, une partie commune à tous les agents SNMP d'un même type de matériel et une partie spécifique à chaque constructeur. Chaque équipement à superviser possède sa propre MIB. Non seulement la structure est normalisée, mais les appellations des diverses rubriques le sont aussi.

Ainsi que le montre l'Illustration 2.10, chaque branche est identifiée de façon unique et non ambiguë. Par exemple, *SysObjectId* est identifié par l'OID numérique 1.3.6.1.2.1.1.2 ou par sa forme littérale *ISO.Org.DOD.Internet.Management.MIB-II.System.SysObjectId*.

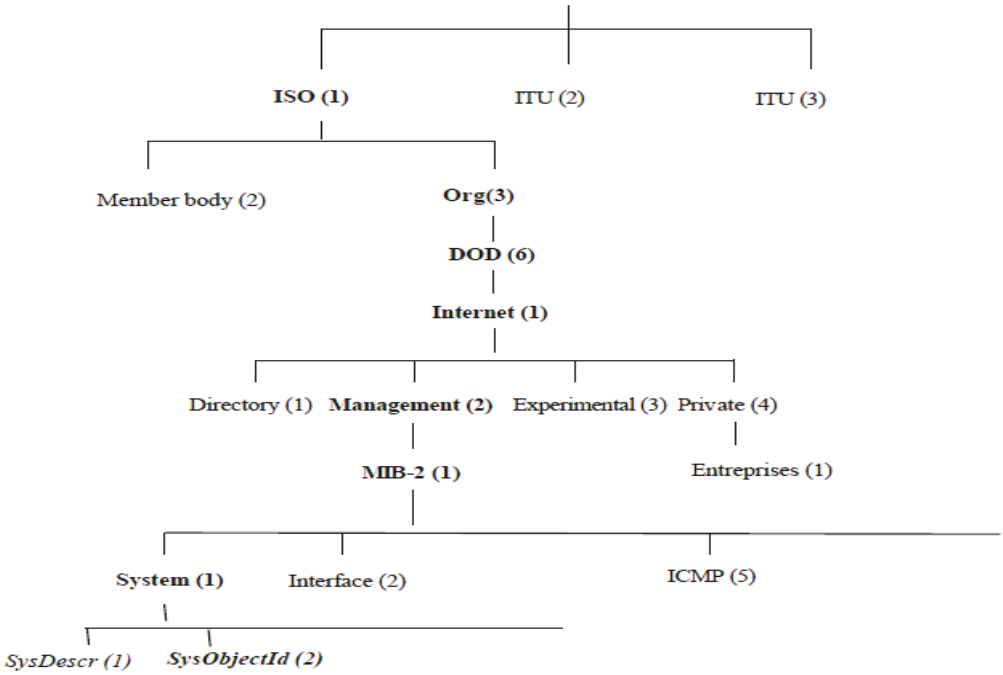


Illustration 2.10 : management information tree : représentation des structures de MIB

La MIB-II correspond à des informations dont disposent tous les équipements compatibles SNMP. Conjointement, les constructeurs de matériel définissent des branches spécifiques avec des informations spécifiques à leurs équipements. L'extrait du Texte 2.1 ci-après illustre une MIB fournie par le fabricant QNAP. Les appellations littérales ne sont présentes que dans un souci de lisibilité. En réalité, chaque niveau de la hiérarchie est repéré par un index numérique et SNMP n'utilise que celui-ci pour y accéder.

```
IfIndex OBJECT-TYPE
    SYNTAX  INTEGER
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            "A unique value for each interface.  Its value
            ranges between 1 and the value of IfNumber.  The
            value for each interface must remain constant at
            least from one re-initialization of the entity's
            network management system to the next re-
            initialization."
    ::= { IfEntry 1 }
IfDescr OBJECT-TYPE
    SYNTAX  DisplayString (SIZE (0..255))
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            "A textual string containing information about the
            interface.  This string should include the name of
            the manufacturer, the product name and the version
            of the hardware interface."
    ::= { IfEntry 2 }
IfPacketsReceived OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            "System packets received."
    ::= { IfEntry 3 }
IfPacketsSent OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            "System packets sent."
    ::= { IfEntry 4 }
```

Texte 2.1: extrait d'une MIB dédiée aux SAN QNAP

Exemple

Les requêtes SNMP sont relativement simples à mettre en œuvre, pour peu que l'on dispose d'une MIB suffisamment explicite. Par exemple, sous les systèmes Linux, on peut disposer de la commande *snmpwalk[32]*, ainsi que l'illustre le Texte 2.2.

Un des intérêts de SNMP, outre sa simplicité et ses performances, est que l'on dispose de multiples outils et API de programmation comme *PySNMP[33]* en langage Python ou Net-SNMP[34] en langage Perl.

```
root@shinken:/# snmpwalk -v 2c -c ghbro  172.16.1.13 If
IF-MIB::ifIndex.1 1
IF-MIB::ifIndex.2 2
IF-MIB::ifIndex.3 3
IF-MIB::ifDescr.1 "lo"
IF-MIB::ifDescr.2 "eth1"
IF-MIB::ifDescr.3 "eth0"
IF-MIB::ifType.1 softwareLoopback
IF-MIB::ifType.2 ethernet-csmacd
IF-MIB::ifType.3 ethernet-csmacd
IF-MIB::ifMtu.1 16436
IF-MIB::ifMtu.2 1500
IF-MIB::ifMtu.3 1500
IF-MIB::ifSpeed.1 10000000
IF-MIB::ifSpeed.2 0
IF-MIB::ifSpeed.3 1000000000
```

Texte 2.2: extrait d'une sortie de commande 'snmpwalk'

2.3.1.2. Le modèle CIM

Si SNMP est le protocole qui semble le plus répandu, cela s'explique aussi sans doute par la vogue de l'environnement TCP/IP. Il existe toutefois d'autres modèles et protocoles qui peuvent être mis en œuvre dans le cadre d'une supervision.

L'approche OSI, par exemple, quoique plus complexe, met en œuvre un modèle de gestion dont les éléments communs sont spécifiés par le service CMIS *(Common Management Information Service element)* et le protocole CMIP *(Common Management Information service Protocol)* que l'on retrouve dans les normes ISO9595 et ISO9596[29]. Quoique plus ambitieuse et généraliste que SNMP, l'approche OSI n'est toutefois pas stabilisée[35]. Cela explique que SNMP soit devenu un standard pour la gestion du « réseau » parallèlement à la version ISO. Toutefois, si SNMP reste simple et performant, il doit être implémenté par les différents constructeurs. Or, si en pratique peu de composants ne supportent pas SNMP, certains fabricants privilégient d'autres solutions et n'enrichissent que très peu les MIB de leurs agents. C'est le cas de Canon, qui fournit des copieurs multifonctions (modèles *iR/Advance*) compatibles SNMP sans procurer les MIB associées aux compteurs spécifiques. En questionnant le service après-vente Canon, il semble acquis que Canon ne fournit jamais ces MIB, mais privilégie des systèmes de supervision sous licence spécifiques à ses produits. Ce type d'approche, qui reste rare, dénote de la possible incomplétude des implémentations d'agents SNMP. Aussi, le fait de s'appuyer uniquement sur SNMP dans le cadre d'un projet de supervision pourrait être un choix fragile.

D'autres outils, souvent propriétaires, sont développés par certaines compagnies. Parmi ces outils, WBEM[36] [37]*(Web-Based Enterprise Management)* a été développé par le *Web Based Enterprise Management Consortium*, qui comprend notamment BMC Software, Cisco, Compaq, Intel, Microsoft, etc. WBEM est une initiative du DMTF *(Distributed Management Task Force)* qui prend en charge le modèle de données CIM *(Common Information Model)*, décrivant les objets d'un environnement de gestion. Le modèle CIM est donc un autre type de modèle. Le DMTF le définit de la façon suivante [38]:

CIM provides a common definition of management information for systems, networks, applications and services, and allows for vendor extensions. CIM's common definitions enable vendors to exchange semantically rich management information between systems throughout the network. CIM is composed of a Specification and a Schema. The Schema provides the actual model descriptions, while the Specification defines the details for integration with other management models.

La page de Wikipedia dédiée à CIM décrit ainsi le modèle[39]:

une autre façon de décrire CIM est de dire qu'il permet à de nombreux intervenants d'échanger des informations de management les concernant. Mais CIM ne permet pas seulement de représenter ces éléments managés ainsi que leurs informations intrinsèques, il fournit aussi un moyen de les contrôler et de les manager directement. En utilisant un modèle de donnée unifié, un logiciel de management peut être écrit une seule fois et travailler ainsi avec plusieurs implémentations de ce modèle unifié, sans surcoût ni perte d'information.

Le modèle CIM se veut plus complexe que le modèle SNMP, avec lequel il permet néanmoins une certaine compatibilité. L'idée est plutôt de proposer un modèle intégré pour certains composants, moins généraliste que le modèle ISO, mais prenant en compte des processus de gestion plus élaborés, en incluant notamment des procédés de contrôle et des API de programmation. Ce que l'on peut comprendre ici, c'est que l'intention de superviser l'intégralité du SI avec ses différentes strates et composants a déjà été formulée au travers de ce modèle. La difficulté avec les implémentations de WBEM, qui s'appuient sur le modèle CIM, est qu'elles ne s'appliquent qu'à certains matériels ou systèmes, offrant finalement peu de prise avec des systèmes d'information très hétérogènes, du fait de cette volonté affirmée d'intégration. D'une certaine façon, si les implémentations de solutions relevant du modèle CIM restent confinées à certains constructeurs, le modèle CIM a l'avantage de fournir une classe d'abstraction et un modèle de représentation unifié des ressources d'un SI. Autre point d'importance : du fait de sa conception, CIM permet d'établir une *relation* entre les composants, ce que ne permet pas SNMP.

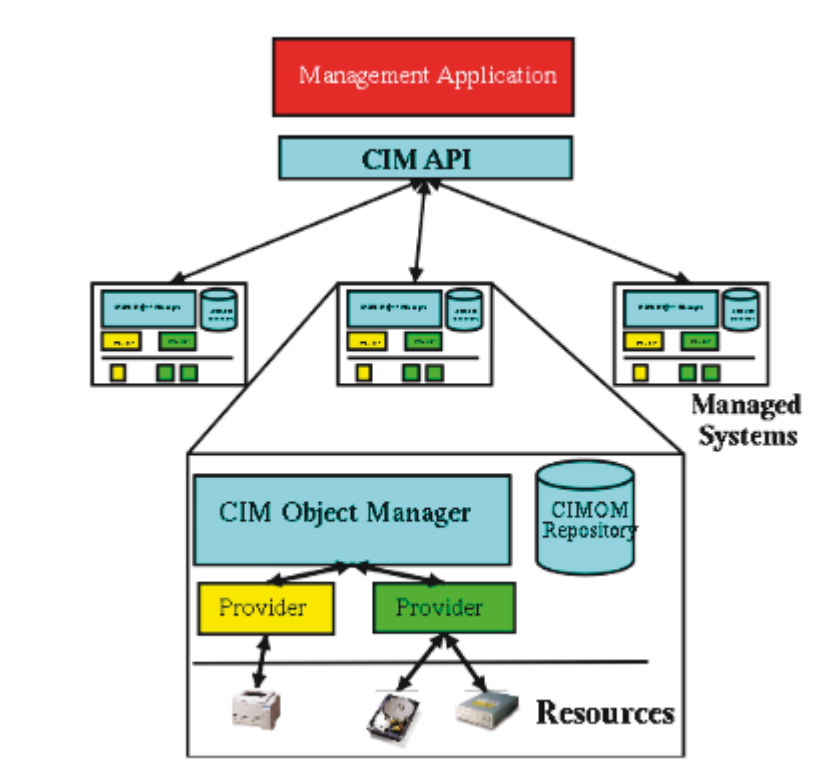


Illustration 2.11 : représentation schématique des interactions entre l'API CIM et les ressources

Outre ces dépendances, le modèle CIM offre une décomposition des éléments d'un SI en s'appuyant sur une modélisation UML. Un méta-modèle standardisé par le DMTF a été défini pour CIM. Ce dernier reprend donc les notions de classe, de propriété, de méthode, d'association et de référence. On distingue notamment :

- les indications qui permettent de représenter la notion d'événement ;
- les qualifieurs qui sont des sortes de « mots-clés » et qui se placent devant une classe, une méthode ou une propriété afin de lui donner une sémantique particulière ou un type particulier ;
- les déclencheurs, ou *triggers,* qui se déclenchent suite à une action.

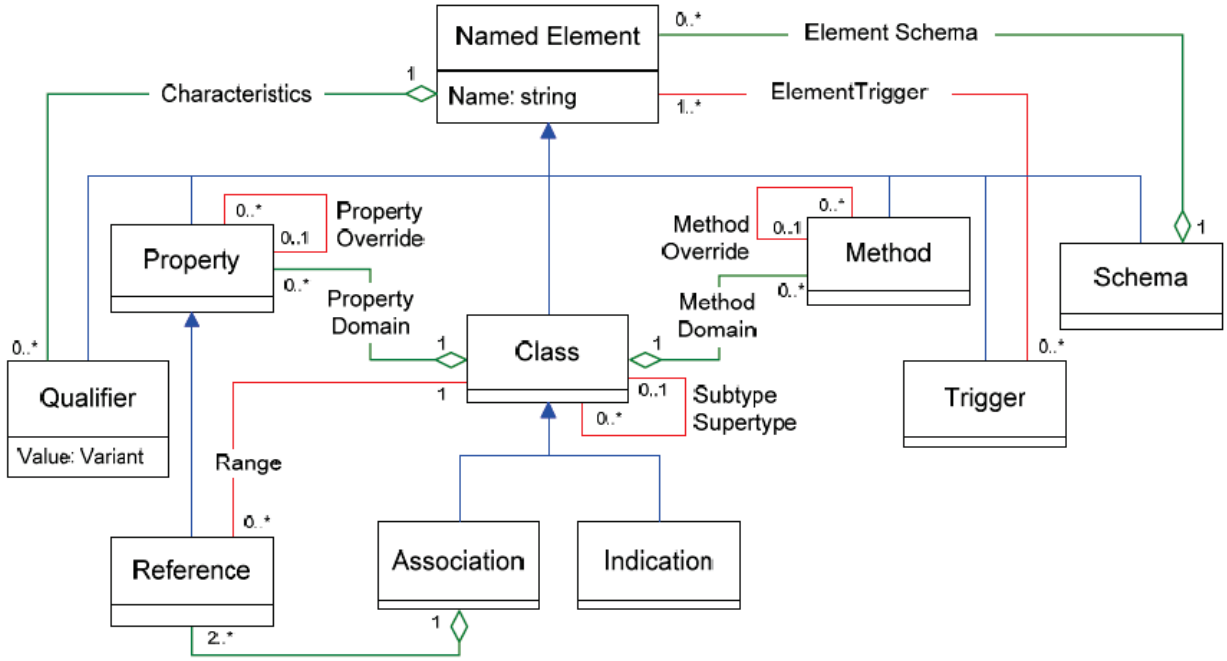


Illustration 2.12 : schéma UML du méta-modèle CIM

À côté de ce méta-modèle sont définis des modèles communs plus génériques. Par exemple, le *Core Schema[38]* distingue les systèmes, les relations entre les composants physiques et logiques, les services, etc. (voir Illustration 2.13). La généricité de ce modèle tend à démontrer l'intention de CIM de s'appliquer à tous les domaines de gestion. Les autres modèles CIM sont des spécialisations du modèle *Core*.

À partir de ce modèle, un certain nombre d'implémentations ont vu le jour, toujours dans l'idée d'offrir une interface de gestion unifiée capable de s'appliquer à l'ensemble des éléments d'un système d'information. C'est le cas de SMI-S *(Storage Management Initiative - Specification)* plus spécifiquement dédié à la gestion des baies de stockage et de WMI *(Windows Management Instrumentation)* qui, lui, s'attache à la gestion des systèmes Windows. L'initiative WBEM, citée plus haut, recense en fait un ensemble de standards de gestion du DMTF, toujours dans l'idée d'uniformiser les interfaces de gestion d'environnement informatique comme SNMP.

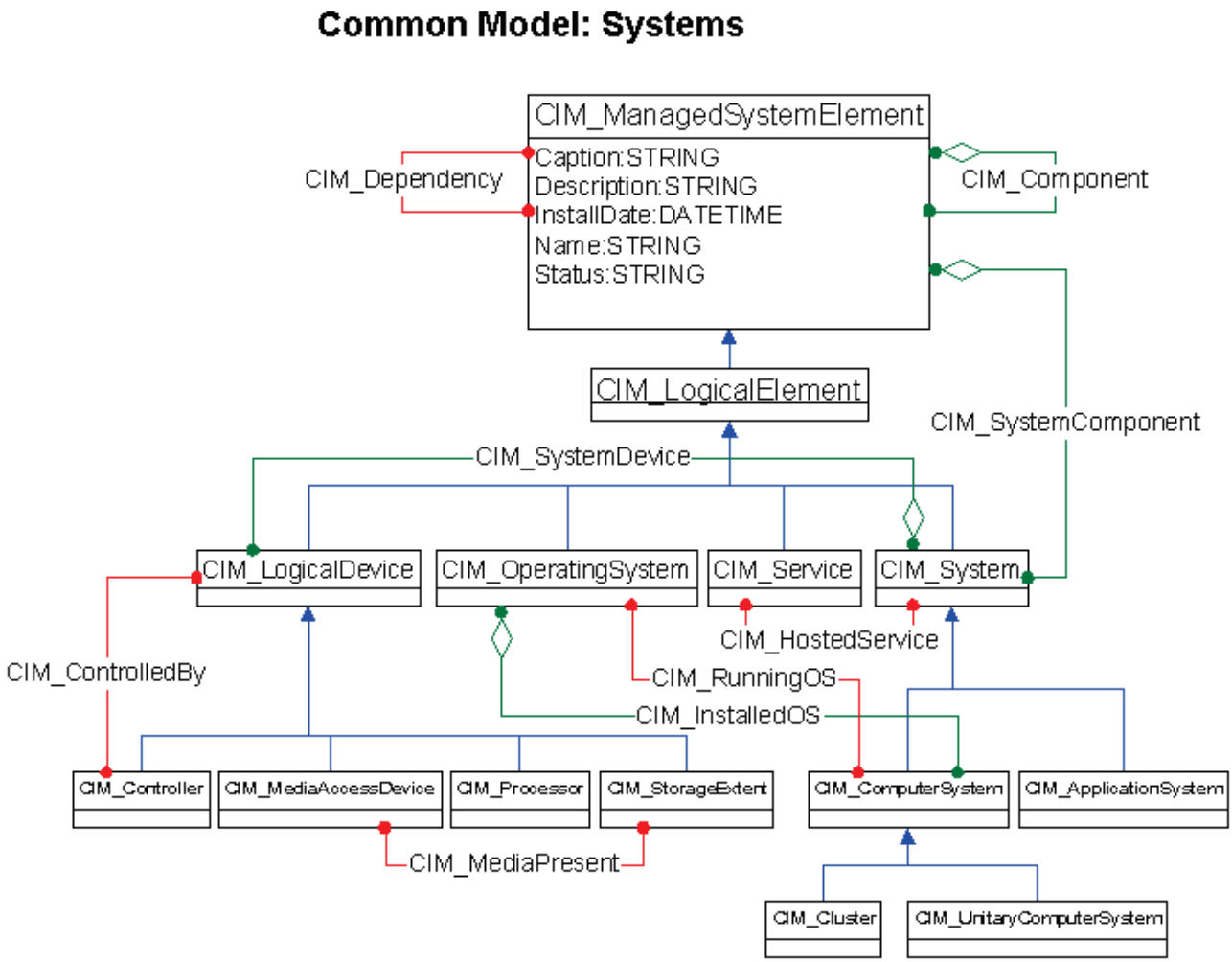


Illustration 2.13 : le modèle CIM « systems »

2.3.1.3. Implémentations reposant sur le modèle CIM

WMI : *Windows Management Instrumentation*

Le WMI est une API Microsoft créée en 1998. Elle vise à permettre la gestion des éléments logiques et physiques de machines dotées de systèmes d'exploitation Microsoft Windows. WMI est en fait une implémentation de la norme WBEM. Le centre de documentation Microsoft définit WMI de la façon suivante[40]:

cette initiative vise à étendre le modèle CIM (Common Information Model) pour représenter les objets de gestion dans les environnements de gestion Windows. Le modèle

CIM, une norme issue également du groupe de travail DMTF, est un modèle de données extensible permettant d'organiser logiquement les objets de gestion, de manière cohérente et unifiée dans un environnement géré.

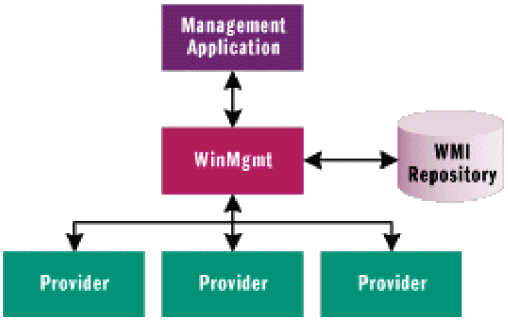


Illustration 2.14 : architecture WMI

Les composants du système sont organisés en classes et objets, à l'instar du modèle CIM. Partant d'une requête, on peut donc agir sur les différents objets d'un système Windows. Soit l'objet est stocké dans le dépôt WMI *(WMI repository* sur l'Illustration 2.14*)*, soit on peut y accéder via un fournisseur, par exemple un *provider* SNMP interfacé avec l'agent WMI.

L'exemple ci-dessous illustre une requête WQL[41] *(Windows Query Langage)* qui récupère la date du dernier démarrage système.

```
Set objWMIService = GetObject("winmgmts:\\.\root\cimv2")
Set colItems = objWMIService.ExecQuery("Select * from Win32_OperatingSystem",,48)
For Each objItem in colItems
    WScript.Echo "LastBootUpTime: "& objItem.LastBootUpTime
Next
```

WMI est évidemment spécifique aux systèmes Microsoft Windows, ce qui en limite la portée. L'implémentation reste toutefois particulièrement intéressante dans le cadre du projet de supervision, notamment parce qu'elle permet d'obtenir des informations très précises sur l'état des composants du système d'exploitation. Du point de vue programmatique et en utilisant les clients adéquats[42], il est relativement aisé d'élaborer des sondes capables de requêter des services WMI distants.

SMI-S

SMI-S *(Storage Management Initiative Specifications)* est un standard dédié au stockage, développé et maintenu par le SNIA *(Storage Networking Industry Association[43])*. SMI-S est également ratifié par un standard ISO (ISO/IEC24775:2011[44]).

SMI-S repose sur CIM et WBEM. L'objectif de SMI-S est de permettre la gestion de systèmes de stockage hétérogènes.

SMI-S définit une série de profils de gestion CIM pour les systèmes de stockage. Ces profils décrivent les différents comportements spécifiques d'éléments comme la gestion des disques RAID, les commutateurs fibre ou réseau, la gestion des volumes, etc. De façon très basique, SMI-S implique deux parties :

- les clients qui requêtent des fournisseurs SMI-S en utilisant WBEM ;
- les serveurs qui sont le plus souvent les périphériques de stockage eux-mêmes (lecteurs LTO, stockage virtualisé, SAN, NAS, etc.). Concrètement, les clients requêtent un serveur CIM qui intègre les profils SMI-S du constructeur concerné.

SMI-S incorpore également des fonctions de découverte via SLP *(Service Location Protocole)*.

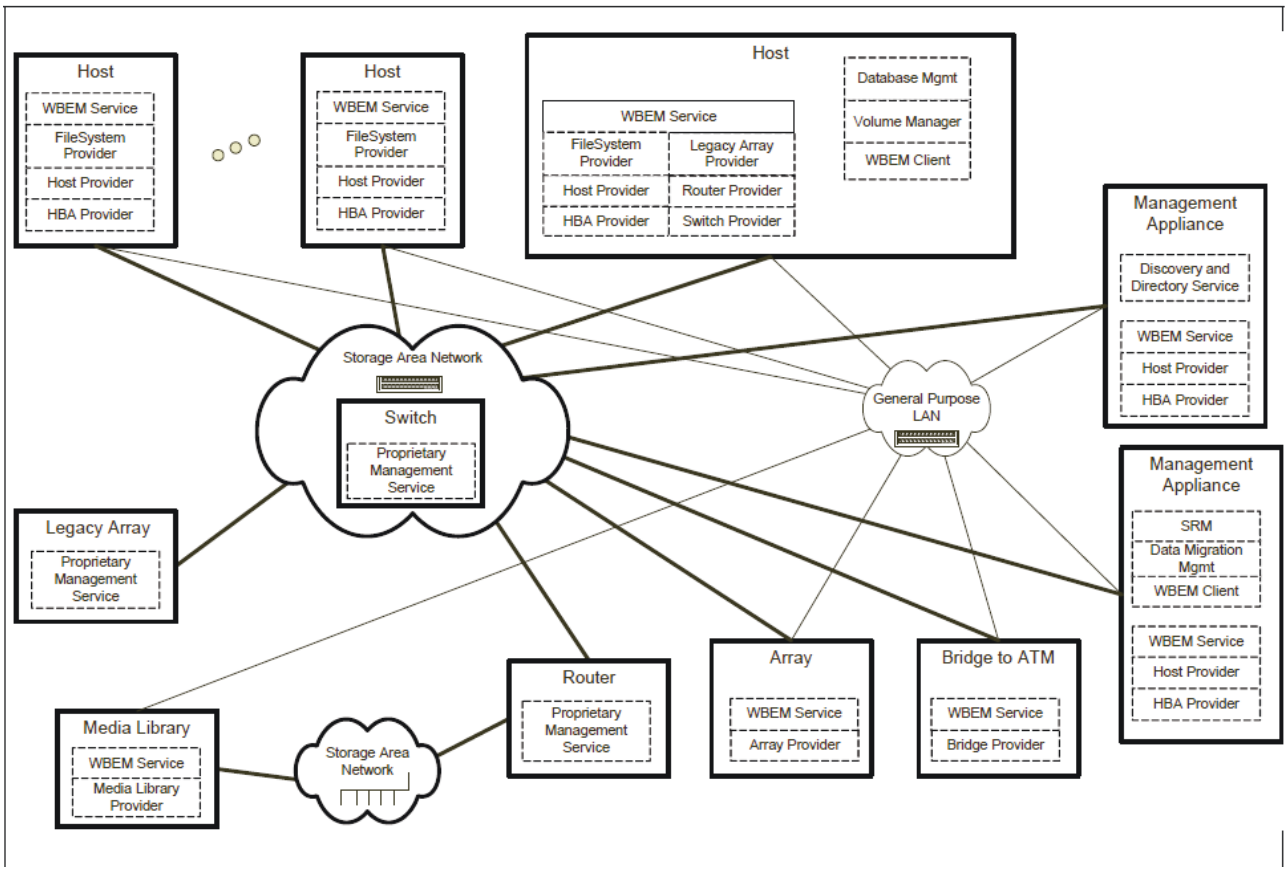


Illustration 2.15 : SMI-s : client - server SAN extrait des spécifications SMI-S

La compatibilité SMI-S ne concerne toutefois que les constructeurs « affiliés ». L'accès aux profils des différents constructeurs est le plus souvent lié à des clients spéci-

fiques, comme par exemple l'utilitaire *smcli*[20] d'IBM ou le client *Navisphere* du constructeur EMC. Cette restriction, parfois incontournable, participe à diversifier les contraintes protocolaires à prendre en compte dans les procédés de supervision.

IPMI

IPMI *(Intelligent Platform Management Interface)* est une interface standardisée utilisée pour la gestion des ordinateurs et leur supervision. IPMI permet d'opérer sur des machines directement, sans passer par le système d'exploitation. Cela autorise notamment d'opérer sur des matériels dont le système d'exploitation est défaillant ou non démarré. Cette initiative est menée par la société Intel et concerne près de 200 vendeurs de machines informatiques[21] dont Cisco, Dell ou NEC.

Les machines dotées de composants matériels implémentant IPMI sont en capacité de renvoyer des informations relatives aux sondes de température interne d'un CPU ou d'un châssis, à la vitesses de ventilateurs, à l'état physique des disques, ou d'envoyer des signaux via des LED. Les systèmes compatibles avec la version 2.0 d'IPMI sont également en capacité de mettre en œuvre des KVM via un réseau LAN, ou d'interfacer les machines distantes avec des média virtuels.

Si IPMI procure des facilités d'administration évidentes - comme l'installation d'un OS ou une modification du BIOS à distance - il s'interface également avec le protocole SNMP. De fait, on peut donc effectuer des requêtes SNMP vers un serveur IPMI qui propose explicitement un support SNMP. Il reste à noter que ce support n'est pas nécessairement complet - certains matériels ne proposent par exemple que la possibilité d'émettre des *traps* SNMP.

En fin de compte, l'intention d'IPMI ne porte pas uniquement sur la question de la supervision, mais plus exactement sur des questions de gestion matérielle. Il reste que ce protocole demeure dans certains cas le moyen exclusif d'accéder à des informations matérielles ; en ce sens, il constitue un outil de supervision non négligeable.

2.3.1.4. Le point de vue utilisateur : End User Experience et Application Performance Management

Face à ces protocoles et normes qui valorisent davantage la technicité du SI, la notion d'expérience utilisateur *(End User Experience)* fait surface depuis quelques années. Elle est l'une des techniques qui compose l'APM *(Application performance Management)[10]*. Quoique peu formalisée, l'APM dispose d'une sorte de modèle (Illustration 2.16) qui recense les différentes techniques utilisées pour évaluer la performance

20 Smcli :http://pic.dhe.ibm.com/infocenter/director/pubs/index.jsp?topic=%2Fcom.ibm.director.cli.helps.doc%2Ffqm0_r_cli_smcli.html

21 IPMI adopters list : http://www.intel.com/content/www/us/en/servers/ipmi/ipmi-adopters-list.html

applicative en valorisant le point de vue utilisateur, plutôt qu'en s'attachant à évaluer les composants de l'infrastructure.

Dans le cas de l'EUE, l'idée est d'évaluer le ressenti d'un utilisateur en simulant son comportement, plutôt que d'évaluer l'ensemble des composants qui concourent à le lui fournir. Il ne s'agit donc pas d'évaluer le ressenti subjectif des utilisateurs réels, mais de vérifier la validité de scénarios comportementaux. Les technologies qui permettent la mise en œuvre de ce type d'évaluation reposent sur des outils de simulation et sur des scénarios fonctionnels découpés en étapes.

Il existe différentes familles d'outils de simulation. On distingue :

- les automates spécialisés dans la simulation web : *watir-webdriver[45]*, *Selenium[46]*, *web-inject[47]* ou même des *frameworks* comme le fameux *WWW::Mechanize[48]* Perl, qui permettent de programmer une interaction avec un navigateur web, ou directement avec des applications en ligne ;
- les outils plus généraux qui reposent sur des techniques d'analyse et de reconnaissance optique. *Sikuli[49]* en est un exemple : ce produit se distingue des précédents automates parce qu'il n'est pas limité à la simulation d'applications web, mais peut être utilisé dans le cadre plus général d'un poste de travail. Son principe de fonctionnement est le suivant : b*y recording both input events and screen images, it is possible to extract the images of components interacted with and the visual feedback seen by the demonstrator, and generate a visual test script automatically.[...] To execute an action statement, the automation engine visually identifies the target GUI component's current location (x!, y!) by searching the current screen for an image region matching the target image I. To find a given pattern, we apply the template matching technique with the normalized correlation coefficient implemented in OpenCV in our current system. This technique treats the pattern as a template and compares the template to each region with the same size in an input image to find the region most similar to the template. Then, the click event is delivered to the center of the matched region to simulate the desired user interaction.[50];*
- on recense enfin diverses techniques permettant de tracer l'exécution de l'application, parfois à l'aide d'un code intrusif. Parmi celles-ci, on peut citer *JMX[51]* pour les technologies Java, le *framework* Javascript *Boomerang[52]* qui permet d'obtenir des informations sur le chargement des pages web dans lequel on l'insère, ou encore des techniques d'analyse reposant sur le *port mirroring*, qui consiste à capturer le trafic d'un commutateur réseau. Certaines de ces techniques sont utilisées de façon intensive dans le cadre de l'automatisation de tests de développement. C'est par exemple le cas de *Jenkins[53]*, un outil d'intégration

continue capable d'automatiser des tests de non-régression. De façon plus générale, on cherche à évaluer une application en traçant l'impact d'un scénario, de bout en bout, sur l'ensemble des composants sollicités.

Illustration 2.16 : APM Conceptual Model [56]

L'APM est une notion encore récente. Son périmètre n'est pas défini de façon stricte : certaines sociétés, comme Compuware[54] ou Capensis[55] mettent en avant cette approche[56], en valorisant un modèle qui distingue les différentes dimensions de l'APM (voir l'Illustration 2.16). Dans le cadre de ce projet, nous nous sommes essentiellement intéressés aux tests comportementaux.

Les procédés basés sur l'analyse syntaxique des pages HTML, comme *WWW::Mechanize[47]* ou *web-inject[47],* ont été abandonnés en raison des limitations de cette technique : les requêtes asynchrones ou la présence d'API tierces comme Microsoft *activeX* ou Adobe *Flash* échappent au procédé d'analyse syntaxique. Qui plus est, de simples modifications d'une page peuvent invalider le code d'une sonde de ce type.

Pour parer à cet écueil, le recours à des solutions plus récentes comme *PhantomJS[57]* ou *Selenium[46]*, qui reposent sur l'émulation complète d'un navigateur, s'est révélé plus efficace. Ces techniques, plus adaptées à la complexité des interfaces web actuelles, permettent de simuler avec plus d'exactitude des scénarios comportementaux liés

à des applications web, notamment en prenant en compte les spécificités liées à l'utilisation du type de navigateur. Ainsi, *Selenium* permet d'émuler différents navigateurs *(Firefox, Internet Explorer).*

Les techniques dites d'APM reposent très largement sur les tests comportementaux. Ce type de test demeure le plus facile à mettre en œuvre, quel que soit le contexte.

Sans couvrir tous les différents aspects énumérés par le modèle de l'APM, il semble pertinent de prendre en compte cette approche comme une composante à part entière des techniques de supervision. Moins centrée sur l'infrastructure, cette approche alternative permet davantage de valoriser les aspects fonctionnels de la supervision.

2.3.2. Le marché des solutions de supervision

Les différents standards et protocoles examinés plus haut fournissent des moyens d'acquisition d'information conséquents. Toutefois, ils souffrent chacun de limitations propres. Par exemple, WMI ne concerne que les systèmes Windows, les implémentations de SNMP différent selon les composants et ne proposent pas une restitution uniforme des informations. SMI ne concerne que certains matériels, aussi son utilisation reste limitée à quelques cas particuliers, comme par exemple les baies de stockage SAN. En fin de compte, il n'est évidemment pas envisageable de s'appuyer sur un standard unique pour superviser une infrastructure hétérogène.

Le choix d'une solution de supervision repose sur des contraintes qui dépassent la seule question du protocole de supervision. De façon à fédérer et à uniformiser les informations du SI et au vu des analyses préalables, les contraintes suivantes ont été identifiées pour caractériser les possibilités d'une solution de supervision :

- support de différents protocoles de gestion : il s'agit de la capacité à s'adapter à un environnement hétérogène où différents protocoles peuvent être mis en œuvre : SNMP, WMI, SMI, etc. ;
- métrologie : capacité à récupérer, historiser et présenter les informations en vue d'analyses ultérieures. Le système de supervision doit également être en mesure de restituer de façon lisible pour un utilisateur non-technicien les informations qu'il collecte. Cela suppose donc une interface de présentation des données spécifique ;
- gestion des relations de dépendance : le système de supervision doit être en capacité de reconstituer les composants en groupes logiques, par exemple pour restituer certaines vues applicatives ou pour représenter un service au sens ITIL ;
- modularité : le SI n'est pas figé, le système de supervision doit pouvoir s'y adap-

ter en continu. Par ailleurs, la supervision est envisagée ici dans une progression ;

- calcul d'indicateurs complexes : le système de supervision doit permettre de mettre en œuvre des calculs de KPI, ou de valoriser des mesures qui ne sont pas fournies directement par les standards ;
- gestion et remontée des alertes, en fonction de seuils ou de KPI ;
- coût d'acquisition moindre ;
- facilité d'administration : la solution de supervision doit présenter des facilités en matière de gestion des configurations ou, de façon plus large, de gestion et d'administration.

Les solutions de supervision sur le marché sont très nombreuses[58]. Globalement, le marché de la supervision est particulièrement atomisé, dans le sens où il comprend de nombreux acteurs spécialistes ou de taille modeste. On distingue généralement les solutions spécialisées dans un domaine spécifique de la supervision comme le réseau, les performances applicatives et la sécurité, mais aussi des solutions plus généralistes qui visent à englober l'ensemble de ces domaines. De fait, les solutions de supervision proposées par les plus importants acteurs du marché, comme HP *(Operation Manager[59])*, IBM *(Tivoli Monitoring[60])* ou Computer Associate *(Nimsoft Monitoring[61])* s'intègrent souvent comme les composants de solutions de gestion d'infrastructure. Hormis leur coût d'acquisition, les applications propriétaires ont l'inconvénient de se présenter comme des *boîtes noires* et posent la question de l'adaptabilité aux spécificités d'un système d'information : le choix de l'outil est clairement lié à une représentation du SI qui, bien qu'encadré par un jeu de protocoles et de modèles, reste difficile à définir de façon normative. En ce sens, les outils propriétaires « clés en main » présentent le risque d'imposer au SI une représentation peu adaptée. En effet, la possibilité de moduler l'outil pour superviser un composant imprévu pose problème lorsqu'elle est soumise à des limitations contractuelles ou techniques.

Pour cette raison, il nous a semblé important de distinguer les solutions payantes, sous licence, des solutions libres ou open-source. Cette seconde catégorie a retenu notre attention, ceci pour trois raisons :

- d'une part, le coût d'acquisition des solutions propriétaires peut être extrêmement élevé, notamment lorsque les besoins de supervision sont amenés à évoluer. Les coûts d'acquisition sont souvent proportionnels au nombre d'agents déployés ou aux fonctionnalités accessibles. C'est le cas de produits comme *PandoraFMS*[22], IBM -Tivoli *NetCool/Omnibus[62]* , ou *POM[63]*;
- d'autre part, les administrateurs système de GBH détiennent une bonne connais-

22 http://pandorafms.com

sance du monde des logiciels libres. Outre les facilités de mise en œuvre que cela implique en termes de connaissance, les solutions libres dont les licences autorisent la manipulation du code présentent des avantages évidents en matière d'adaptation et de modularité.

- enfin, la transparence d'un procédé de remontée d'alertes est fondamentale pour un administrateur système : on ne peut pas évaluer la gravité d'un problème ou faciliter sa résolution sans maîtriser les modalités de cette évaluation.

Dans le monde open-source et libre, on trouve également un grand nombre de projets ayant la supervision comme objet. On distingue :

- des solutions « tout-en-un » qui intègrent les mécanismes de supervision : sondes, remontées d'alertes, métrologie, visualisation des données, etc. C'est par exemple le cas de *Zabbix[64]*, *OpenNMS[65]* ou *Centreon[66]*. Toutefois, ces solutions restent peu modulaires et la gestion des dépendances, sauf dans le cas de *Centreon*, y est peu développée ;
- des outils spécialisés en métrologie :
  - *Cacti[67]*, très orienté réseau et métrologie et qui repose essentiellement sur SNMP,
  - *Graphite[68], rrdtools[69]* qui sont des outils dédiés aux graphes,
  - *Nagvis[70]*, une solution graphique plus spécifiquement orientée cartographie,
  - Ganglia[71], qui est dédié à la métrologie des environnements de type *cluster ;*
- des solutions plus modulaires où les sondes, les interfaces de monitoring et les logiques de surveillance et de contrôle s'agrègent au gré des besoins. Il s'agit pour l'essentiel de *Nagios* et de toutes les solutions affiliées comme *Shinken*, *Icinga*, *EyesOfNetwork* ou *Centreon* ;
- des solutions dites d*'hypervision*, sur lesquelles nous reviendrons plus tard et qui visent à agréger et à consolider des informations et des événements provenant de sources diverses (remontées de sonde, *Syslog*, métriques hétérogènes, etc.). Des produits comme *Canopsis[72]* ou *Opinsight[73]* se situent dans cette lignée. Ces solutions interviennent en amont de la récolte de données de supervision et présentent un intérêt évident en matière de présentation et de visualisation des données ;

Plusieurs critères et exigences ont permis d'orienter le choix de la solution :

- les performances : les procédés de supervision ne doivent pas interférer de façon

significative avec les éléments qu'ils examinent. De façon très claire, la supervision ne doit pas créer une surcharge d'activité pour les autres éléments du SI ;

- la gestion de la configuration ne doit pas être trop complexe, de façon à pouvoir évoluer et être maintenue facilement ;
- les sondes de supervision doivent proposer un code lisible et modifiable. Par exemple, si le modèle SNMP autorise les éditeurs à modifier les branches de leurs propres MIB, il faut en contrepartie pouvoir adapter les sondes aux spécificités du SI de GBH ;
- la gestion des dépendances : il doit être possible de restituer les liens de dépendance virtuels ou physiques entre les différents composants supervisés. Si des modèles comme CIM proposent d'emblée une vision des dépendances entre composants, il s'agit toutefois d'un point de vue générique. La possibilité d'établir des liens logiques entre les différentes couches du SI - par exemple représenter un service au sens ITIL en regroupant des composants logiques et physiques hétérogènes - demeure importante ;
- la supervision doit pouvoir produire des résultats lisibles, non seulement pour les administrateurs système, mais également pour les autres utilisateurs du SI. Ce critère renvoie à la question de la représentation des données ;
- la maturité et le dynamisme de la solution constitue enfin un motif d'appréciation important lorsqu'on évalue une solution libre.

Le nombre de solutions libres ou open-source est, comme le marché des solutions propriétaires, extrêmement fragmenté[74]. On en trouve divers inventaires relativement exhaustifs[75] ou des comparatifs[76], qui témoignent de l'enthousiasme que peut nourrir la question de la supervision. À fonctionnalités identiques, il demeure toutefois difficile d'établir une véritable distinction. À ce niveau, la question de la maturité du projet et sans doute aussi celle de sa popularité entrent en ligne de compte comme des éléments d'appréciation non négligeables.

En termes de maturité et de popularité, le projet *Nagios[77]* se dégage vraisemblablement du lot. Il dispose de capacités de modularité suffisamment avancées pour qu'un véritable écosystème applicatif se soit constitué autour de lui. Il dispose en outre d'une solide assise communautaire[78] et de capacités de configuration très avancées. *Nagios* constitue en effet une véritable référence dans le monde de la supervision open-source ; preuve en est des différentes solutions « tout-en-un » qui l'intègrent *(Centreon*, *EyesOfNetwork)* ou de projets affiliés, dits *forks*, comme *Icinga*, *Centreon Engine* et *Naemon* ou *Shinken* - celui-ci n'étant pas véritablement un *fork*, mais plutôt une preuve de concept qui hérite des mécanismes de *Nagios*.

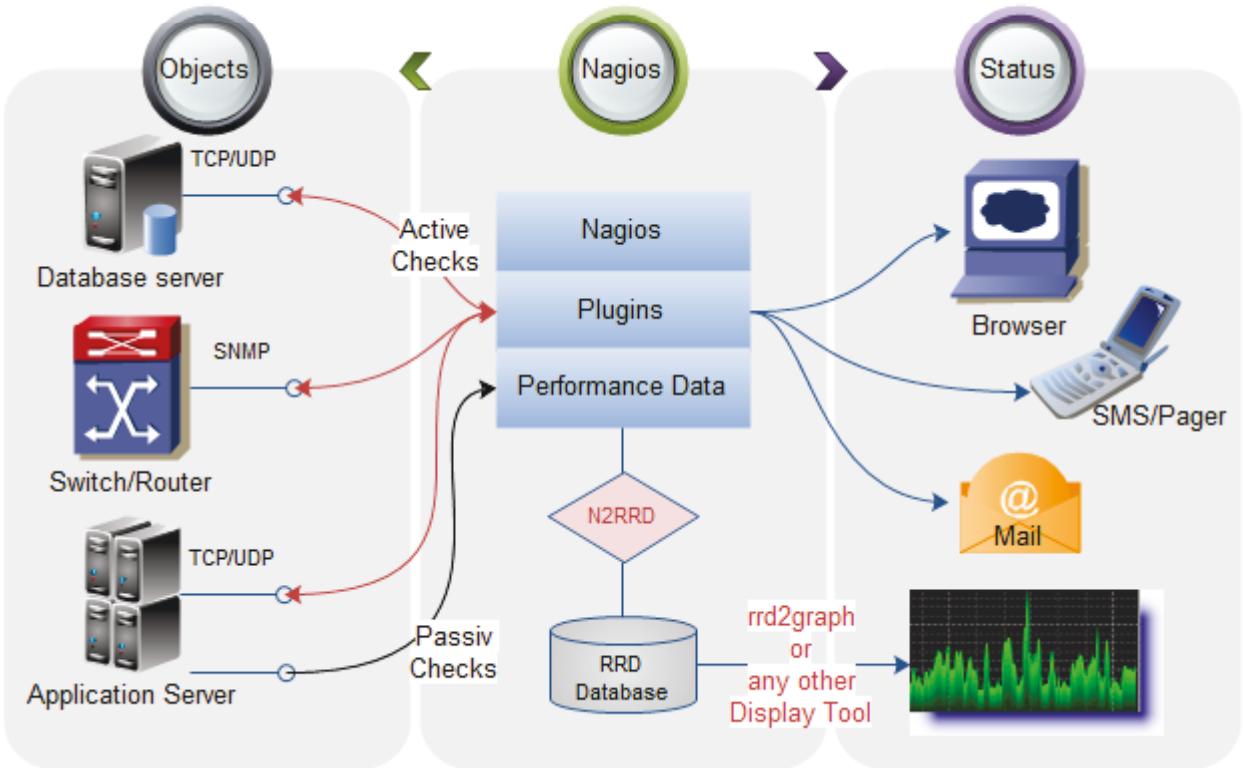


Illustration 2.17 : fonctionnement de Nagios[127]

De façon concrète, *Nagios* n'est autre qu'un ordonnanceur sophistiqué de vérifications, capable de déclencher des alertes ou des actions. Il s'interface avec des plug-ins et des applications tierces qui lui permettent de compléter ses fonctionnalités.

Nagios a initialement été développé sous le nom de *Netsaint* par Ethan Galstad en 1999. Un problème de propriété intellectuelle motive le renommage du projet en *Nagios* en 2002. En 2006, le projet *NDOutils* permet d'intégrer les données de *Nagios* dans une base de données. C'est en s'appuyant sur *NDOutils* que d'autres projets, comme *Centreon[66]* ou *OP5Monitor[79]* se développent, pointant par ailleurs la complexité de *Nagios*. En 2007 apparaît la société *Nagios Enterprise*, symptôme d'une discorde entre la société *Netways* et l'auteur de *Nagios* à propos de l'utilisation du nom. Le dialogue entre la communauté d'utilisateurs et l'auteur de *Nagios* se détériore d'année en année, stigmatisant l'opposition des deux optiques, celle de la société *Nagios Enterprise* qui vise à protéger des intérêts commerciaux et celle de la communauté d'utilisateurs, dont le dynamisme a largement contribué au projet. La naissance de *forks* comme *Icinga* témoigne des difficultés d'évolution du projet *Nagios* qui, bien qu'il en soit aujourd'hui à sa version 4, laisse sceptique quand à sa pérennité.

Le projet *Nagios*, malgré ses assises communautaires, souffre en effet de plusieurs critiques. D'une part, *Nagios* privilégie une approche « technique » *(IT Monitoring)*. Or, le point de vue purement technique reste limitatif : la seule supervision d'une infrastructure ne permet pas d'établir un lien avec les perspectives stratégiques d'une organisation. D'autre part, la gestion de la configuration de *Nagios* reste complexe, ce qui peut devenir un frein évident à sa mise en place et à son évolution. Par ailleurs, on reproche à *Nagios* certaines faiblesses liées à sa conception[80], notamment en ce qui concerne sa gestion interne des processus, qui impactent notablement ses performances lorsque que le nombre d'objets à surveiller s'élève. Enfin, comme évoqué plus haut, la communauté *Nagios* souffre de dissensions[81] qui laissent planer une certaine confusion quant à l'évolution du produit.

Au vu de ces critiques et des objectifs du projet de supervision, nous nous sommes davantage intéressés à des projets affiliés. Parmi eux, le projet *Shinken[82]* comprend un certain nombre d'améliorations et de propositions. Nous observerons celles-ci dans la seconde partie. Un des intérêts évidents de *Shinken* est qu'il permet de bénéficier de l'héritage communautaire du projet *Nagios*, ce qui se concrétise essentiellement par le code de milliers de *plug-ins* de supervision. En effet, *Shinken* bénéficie d'un certain dynamisme, visible entre autres par l'activité de son dépôt *git[83]*.

Au-delà de la mécanique brute de surveillance, il faut également considérer la question de la collecte des données de supervision et de leur présentation. Si *Nagios*, *Shinken* ou *Icinga* disposent de leurs propres interfaces graphiques, il est également possible de les interfacer à d'autres produits, spécialisés dans le rendu visuel. Nous avons déjà cité certains d'entre eux, mais la question de fond vis-à-vis des représentations graphiques est fondamentalement liée aux aspects stratégiques et métier du SI. La prise en compte de ces derniers nécessite que les tests de vérification puissent prendre sens pour les utilisateurs profanes.

3.Principes de l'architecture cible

3.1. Shinken

Shinken a été écrit par Jean Gabès. Initialement conçu comme une *preuve de concept* visant à améliorer les défauts de performances de *Nagios*, *Shinken* est un projet relativement jeune et dynamique qui propose une solution de supervision sans doute aussi modulaire que *Nagios* et dont nous examinerons ici l'architecture générale.

3.1.1. L'héritage de *Nagios*

Le projet *Nagios* évoqué plus haut a, de par son succès, quasiment instauré une méthodologie technique de supervision. Aussi est-il important d'en reprendre certains points pour comprendre l'héritage de *Shinken*.

3.1.1.1. Méthodes d'obtention des informations

Ainsi que nous l'avons noté, la solution *Nagios* est fondamentalement un ordonnanceur spécialisé dans l'exécution de sondes, la récupération de leurs résultats et le déclenchement d'alertes.

Il existe deux méthodes de récupération des informations

- méthode active : elle consiste à laisser l'initiative de l'exécution de la sonde à l'ordonnanceur ;
- méthode passive : elle consiste à émettre des informations, alertes ou résultats d'une sonde par une instance externe à l'ordonnanceur. Il s'agit typiquement de faire récupérer des instructions SNMP de type *trap* par l'instance *Nagios*.

Certains programmes qualifiés d'*add-ons* facilitent l'exécution de commandes de vérification sur des hôtes délocalisés. Il s'agit pour l'essentiel de NRPE *(Nagios Remote Program Executer[84])* ou NSCA *(Nagios Service Check Acceptance [85])*.

Les commandes de vérification sont des programmes exécutables capables de récupérer des informations sur une instance locale ou distante. C'est à ce niveau que l'on sollicite différents protocoles de gestion comme SNMP ou WMI. Ces programmes sont aussi désignés sous le terme de plug-ins.

3.1.1.2. Commandes de vérification (checks)

Les commandes de vérification doivent respecter une trame générale dont les spécifications sont détaillées dans *Nagios Plugin Development Guidelines*[23]. Aucun langage ou méthode d'accès n'est imposé par *Nagios*. Seule la sortie *(output)*, soit le format des informations renvoyées par la sonde, doit être compréhensible par l'ordonnanceur. Ces *checks* représentent n'importe quelle commande capable de restituer des informations pertinentes pour la supervision.

3.1.1.3. Le « modèle » Nagios

Nagios fait la distinction entre les hôtes, également nommés nœuds, qui représentent le plus souvent un serveur doté d'une adresse IP qui lui est propre, et les services, qui sont des groupes logiques regroupant les facultés de l'hôte. La notion de service ne correspond donc pas à celle qui définit les processus d'un système. Elle correspond à une définition plus large qui peut englober aussi bien des composants logiques que physiques.

Illustration 3.1 : les relations des objets de type service dans Nagios[86]

Les objets *Nagios* suivent un schéma d'héritage spécifique. Les liens entre les objets sont restitués dans les fichiers de configuration. Ci-dessus, le schéma simplifié, extrait de l'ouvrage *Nagios 3 Enterprise Network Monitoring: Including Plug-Ins and Hardware Devices[86]* reconstitue ce modèle.

23 *Nagios Plugin Development Guidelines* : https://nagios-plugins.org/doc/guidelines.htm

La principale difficulté de ce modèle est sans doute sa généricité : moins précis que le modèle CIM, il est conçu pour s'adapter à un grand nombre de situations. Ce modèle met également en œuvre une notion d'héritage à travers un système de modèles *(templates)*. Cet héritage, qui est astreint à des règles de précédence dans les cas d'héritages multiples, constitue une des difficultés notables de la mise en œuvre d'une configuration *Nagios*[24]. L'idée étant d'adapter le modèle générique de *Nagios* aux spécificités d'un SI, il faut nécessairement, pour des raisons de lisibilité évidentes, mettre à profit des modèles de configuration. Typiquement, des systèmes d'exploitation de la même famille, comme différents systèmes Linux, peuvent présenter des caractéristiques qui s'analysent par les mêmes procédés. Dans le cas de systèmes Windows, le recours à WMI peut par exemple être systématisé en s'appuyant sur un de ces modèles.

3.1.1.4. Statut des objets

À la suite du test qu'il effectue, le *plug-in* doit pouvoir renvoyer un statut. *Nagios* s'appuie sur la nomenclature Unix pour définir ces codes. Pour un objet de type hôte, la correspondance des codes retour des programmes avec leur signification est explicitée dans le Tableau 3.1 :

Code retour	Statut	Description
0	UP	L'hôte est actif sur le réseau
1	UP	L'hôte est actif si la vérification a été forcée
2 ou 3	DOWN	L'hôte n'est pas joignable par le réseau

Tableau 3.1 : correspondance entre le code retour d'un plu-gin et l'état d'un hôte

Lorsque l'objet est un « service », la signification est plus nuancée. Elle suit la nomenclature explicitée dans le Tableau 3.2 :

24 *Nagios objects inheritance* : http://doc.monitoring-fr.org/3_0/html/advancedtopics-objectinheritance.html

Code Retour	Statut	Description
0	OK	Statut normal
1	WARNING	Avertissement, un problème non-critique a été détecté, comme par exemple le dépassement d'un seuil de remplissage d'un disque
2	CRITICAL	Statut critique situation grave demandant une intervention immédiate
3	UNKNOWN	La commande n'a renvoyé aucune information

Tableau 3.2 : correspondance entre le code retour d'un plug-in et de l'état d'un service

Cette première nomenclature fournit une indication générale de l'état d'un objet. Ce code retour peut être accompagné d'un texte d'information et de données de performance. Le format général de la sortie d'une commande de vérification est donc :

```
Service Status Code : Information texte | |'label'=value[UOM];[warn];[crit];[min];[max]
```

Par exemple, la sortie suivante :

DISK OK - free space: / 3326 MB (56%); |/=2643MB;5000;7000 ;0;7000

indiquerait un statut normal *(OK)* concernant une partition racine occupée à 44 % par un volume de 2643Mo. Les données de performance indiquent par ailleurs qu'un warning serait déclenché dès le dépassement de 5000 Mo d'occupation et qu'une alerte critique serait émise pour 7000 Mo

Au final, le code d'un *plu-gin* doit comprendre trois actions :

1. la vérification *(check)* ;
2. l'affichage de la sortie, qui comprend éventuellement les détails littéraux et les données de performance ;
3. le code retour du programme.

L'écriture de *plugins Nagios* peut être facilitée par certains outils. Le module Perl *Nagios::Plugin*[25] en est un exemple, mais il en existe d'autres comme *PySNMP*[26] qui permettent de travailler sur les objets SNMP en langage Python, ou *Check_wmi_plus*[27], qui s'attache à simplifier les requêtes WMI et le traitement des informations retournées par les hôtes Windows.

25 *Nagios::Plugin* est un module du CPAN : http://search.cpan.org/dist/Nagios-Plugin/lib/Nagios/Plugin.pm

26 *PySNMP* : http://pysnmp.sourceforge.net/

27 http://www.edcint.co.nz/checkwmiplus/

3.1.1.5. Ordonnancement des vérifications

Le lancement de ces commandes par l'ordonnanceur est soumis à des règles de configuration qui sont très semblables dans *Skinken*.

À titre d'exemple, l'état d'un « service » testé évolue selon des intervalles de temps prédéfinis. On observe dans l'Illustration 3.2 l'effet des éléments du paramétrage suivant :

```
max_check_ attempt 4
check_interval 5
retry_interval 3
```

Illustration 3.2 : évolution de l'état d'un service en fonction des résultats des sondes

Cela signifie que l'on effectue une vérification toutes les 5 minutes *(check_interval)*. Si le résultat du test n'est pas le statut 'OK', alors on effectue 3 essais supplémentaires. Le statut du service testé passe alors de l'état *soft* à l'état *hard*. À la suite de quoi, les tests reprennent en fonction du paramètre *check_interval*.

3.1.1.6. Synthèse

Globalement, il faut établir les paramètres de configuration des hôtes et des services et leur associer des commandes de vérification. *Nagios* planifie les tests et la récolte des informations en fonction de ces derniers. La récolte d'informations permet notamment d'établir des graphiques de supervision. Le statut des sondes conditionne l'envoi de notifications (mail, sms) ou déclenche d'autres actions *(Event Handler)*. L'Illustration 3.3 représente le schéma fonctionnel de *Nagios*. On y distingue les différents éléments évoqués précédemment.

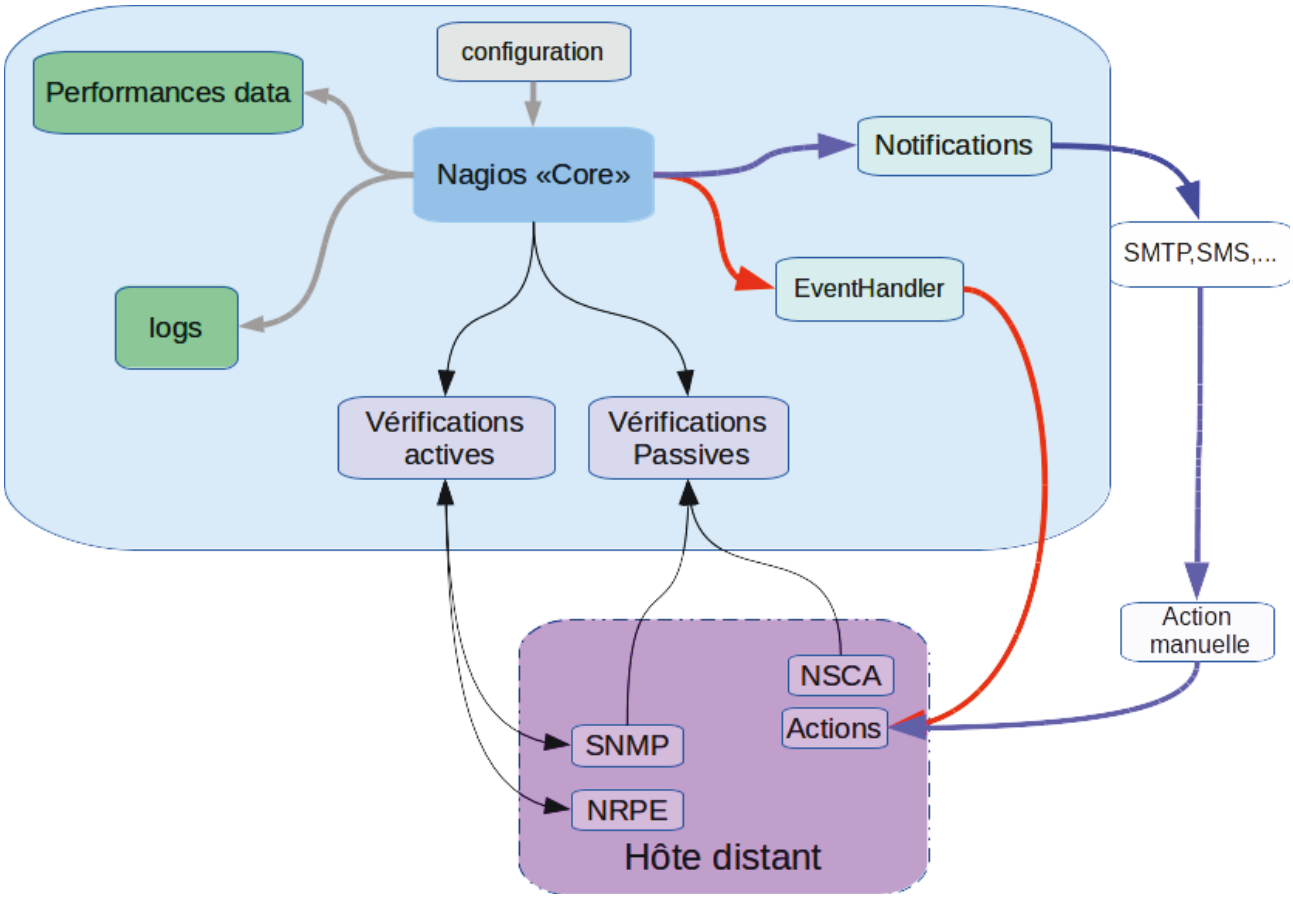


Illustration 3.3 : schéma fonctionnel de Nagios

Dans les grandes lignes, on retrouve les objectifs de la supervision technique d'une infrastructure :

- on organise un jeu de sondes techniques en s'appuyant sur différents protocoles et outils ;
- *Nagios* prend en charge l'ordonnancement de ces vérifications ;
- il en consigne les résultats *(log)* ou les données de performance ;
- à la suite de quoi il peut émettre une alerte ou lancer une action automatique sur l'hôte concerné.

Nagios instaure donc une logique de surveillance et de contrôle qui peut contribuer à la gestion des ressources physiques, à l'évaluation d'indicateurs de performance ou à la détection d'incidents. Il doit toutefois s'entourer d'autres outils – ceux-là mêmes qui forment son écosystème - pour aller plus loin, puisque son rôle reste fondamentalement confiné à l'ordonnancement des tâches.

3.1.2. Architecture générale de *Shinken*

Pour résumer, l'outil *Nagios* fournit les bases d'une architecture de supervision qui repose sur l'exploitation d'un format de message unifié relativement simple. Toutefois, au vu des difficultés d'évolution du projet, il est apparu plus sage de porter l'attention sur un de ses descendants. *Shinken* en fait partie et s'appuie sur une série de constats qui pointent certaines insuffisances de *Nagios* :

- d'une part sa configuration est complexe, mais sa structuration reste monolithique : elle induit une difficulté de maintenance qui s'accroît avec le nombre d'hôtes et de services à superviser ;
- d'autre part, *Nagios* fournit une analyse éminemment technique du système d'information. Or, nous cherchons justement à pouvoir valoriser le lien entre l'infrastructure technique et les attentes stratégiques de l'organisation ;
- *Nagios* peut rapidement poser des problèmes de performances. S'il existe bien des préconisations précises pour le réglage des performances [87], un de ses principaux défauts est lié à sa conception. C'est du moins ainsi que J.Gabès, l'auteur de *Shinken*, présente la situation : *« Le problème de performances est dû au mécanisme qu'utilise Nagios pour obtenir ses informations : il se forke afin que son fils lance lui-même une sonde (un simple script shell ou un exécutable) qui réalise la vérification. Le processus fils lit la sortie standard de la sonde et son code retour pour obtenir l'état de l'élément surveillé. Ce fils écrit ces résultats dans un fichier plat, qui sera lu et parsé par le démon principal. Une fois le fichier écrit, le fils meurt. Dans ces deux dernières phrases réside le cœur du problème de performances de Nagios:*

  - le parsage de fichiers plats est une opération extrêmement coûteuse,

  - pourquoi tuer le processus fils après son travail si c'est pour le relancer à la prochaine sonde ? Un fork est une opération coûteuse. »[88].

L'architecture de *Shinken* se pose initialement comme une preuve de concept visant à revoir l'architecture monolithique de *Nagios*. Implémenté en Python, *Shinken* propose une solution compatible avec *Nagios*, mais découpée en 6 processus autonomes d'arrière-plan, appelés daemon dans la terminologie UNIX. L'Illustration 3.4, extraite du site monitoring.fr[28], schématise cette architecture.

28 Voir plus spécifiquement : http://wiki.monitoring-fr.org/shinken/shinken-work

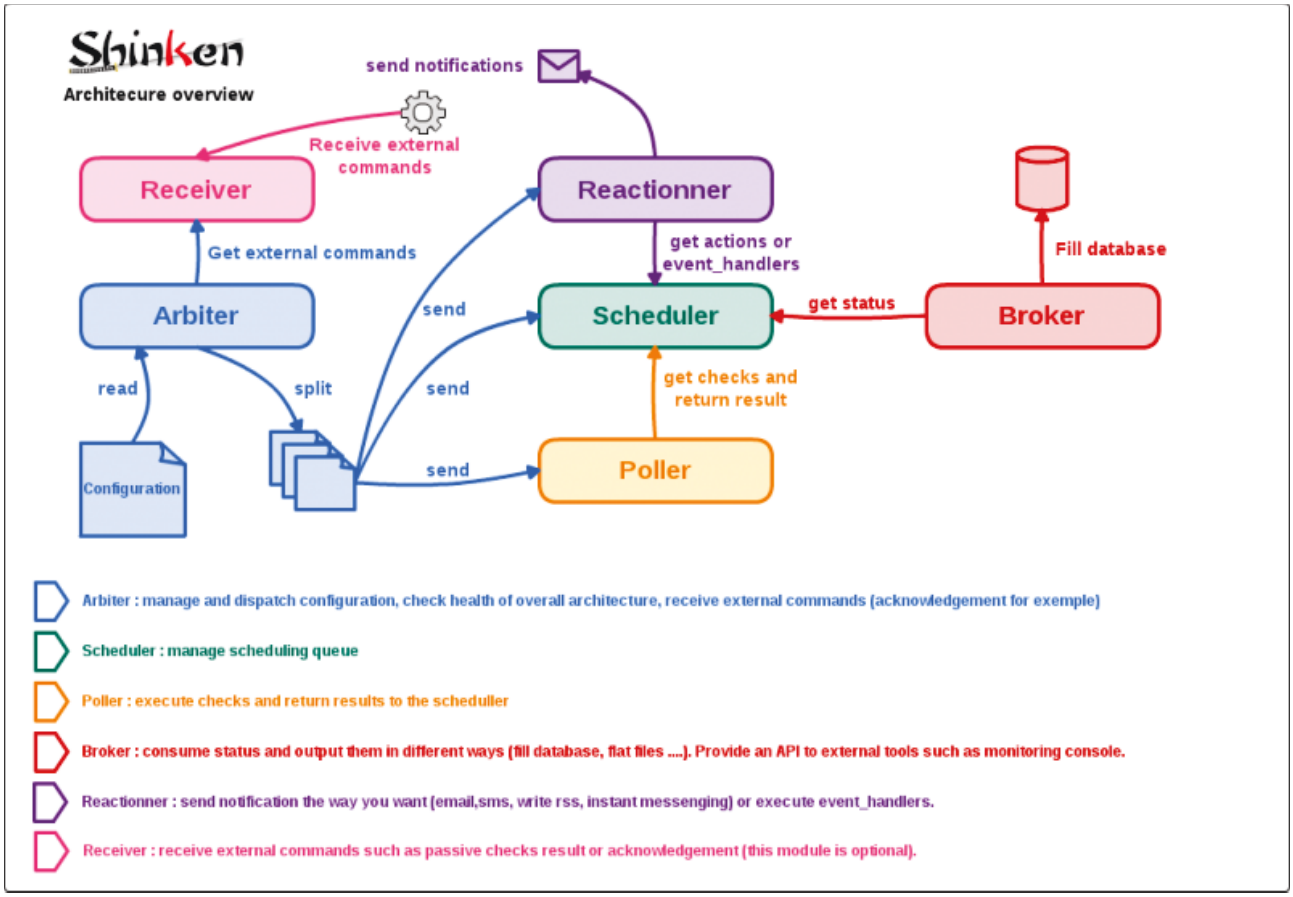


Illustration 3.4 : architecture générale de Shinken,

- le *daemon Arbiter* lit la configuration, la découpe et la distribue aux autres *daemons* ;

  le *daemon Scheduler* est chargé de l'ordonnancement des tâches, de l'analyse des résultats et du déclenchement des actions. Il ne fait que transmettre les informations au *daemon* approprié à la façon d'un routeur ;

- le *daemon Poller* lance les tâches de vérification, les *plug-ins*, en fonction des requêtes du ou des *Schedulers* ;
- le *daemon Receiver* est chargé de traiter les données d'acquisition passives et de les retourner au *Scheduler* responsable des traitements ;
- le *daemon Reactionner* s'occupe de l'envoi des notifications et des actions automatiques programmables ;
- Le *daemon Broker* est chargé de rendre disponibles les données de supervision. *Shinken* propose par ailleurs un certain nombre de modules capables d'exporter les données vers d'autres programmes, API ou bases de données.

On retrouve globalement les mêmes fonctions que *Nagios*, à la différence que dans *Shinken*, chaque fonction est prise en charge par un *daemon* autonome. Les *daemons* communiquent entre eux par l'intermédiaire d'une API spécifique, *Pyro*[29], qui permet aux objets Python de communiquer entre eux. Une des conséquences de cette conception est de permettre l'exécution de *daemons* distants et d'envisager une architecture répartie.

3.1.2.1. Configuration

La configuration de *Shinken* observe un découpage qui s'inspire de *Nagios* en beaucoup plus segmenté. On y distingue essentiellement les ressources et des objets. Les ressources[30] concernent la définition des différents *daemons* et de leurs attributs, ou encore des éléments de définition globaux, comme par exemple la communauté SNMP ou des identifiants de connexion. Les objets[31], quant à eux, définissent les éléments à surveiller : adresse, dénomination, type de test.

À partir de là, il est possible de construire des variables - ou *macros* - dont certaines sont prédéfinies[32]. Ces dernières permettent de propager des valeurs dans la définition d'objets, du niveau global au niveau local. Plus concrètement, cela permet de mettre en œuvre une notion d'héritage entre les objets[33], en créant des modèles d'objets *(templates)*.

Ces définitions s'organisent dans une arborescence de petits fichiers de configuration. L'exemple ci-dessous représente l'arborescence du répertoire de configuration de la version 1.4 de *Shinken*.

```
├── brokerd.ini      //paramètrage du daemon « broker »
├── commands.cfg     // définition de commandes de vérification génériques
├── contactgroups.cfg  // définition des groupes de contact
├── contacts.cfg       //définition des contacts : mail, numéro de tel.
├── dependencies.cfg    //définition des dépendances
[…]
├── escalations.cfg     //définition des modalités de notification
├── hostd.cfg
├── hostgroups.cfg      //définition des groupes d'hôtes
├── hosts               // répertoire de définition des hôtes
│   ├── 172.16.1.14.cfg  // définition d'un hôte
│   ├── 172.16.1.24.cfg
│   ├── 172.16.16.113.cfg
│   ├── 172.16.16.115.cfg
│   ├── 172.16.16.99.cfg
│   ├── 172.16.2.1.cfg
│   ├── 172.16.2.4.cfg
│   ├── 172.16.2.5.cfg
[…]
├── templates.cfg      //définition des modèles
```

29 http://pythonhosted.org/Pyro4/

30 http://www.shinken-monitoring.org/wiki/official/configuringshinken-configmain#configuringshinken-configmain-resource_file

31 http://www.shinken-monitoring.org/wiki/official/configuringshinken-configobject

32 http://www.shinken-monitoring.org/wiki/official/thebasics-macrolist?s[]=macro

33 http://www.shinken-monitoring.org/wiki/official/advancedtopics-objectinheritance

```
├── timeperiods.cfg     //définition des périodes horaires d'activité
├── time_templates.cfg  // modèle pour la définition d'horaires
└── triggers.d          //définition des actions à déclencher
    ├── eue.trig
    ├── imp_counter.trig
    ├── vmfs_pc.trig
    └── vmfs.trig
```

Chacun de ces fichiers définit un ou plusieurs objets en s'appuyant sur les *templates*. Les principaux objets qui peuvent être définis sont :

- les s*ervices* qui sont des objets que l'on associe à un hôte *(host)*. Il peut s'agir d'attributs physiques (mémoire, disque) ou de services logiques (HTTP, DNS, LDAP, etc.) ;
- les *service groups* qui sont des regroupements de services ;
- les *hosts* qui sont des entités possédant un adressage propre sur le réseau. On peut leur associer un ou plusieurs *services*. Les hôtes peuvent entretenir des relations de parenté *(parent/child relationship)* avec d'autres *hosts* ;
- les *host group*s, des regroupements d'hôtes ;
- les *contacts* qui représentent les personnes qui doivent être averties par les processus de notification ;
- les *contact groups* qui sont des regroupements d'objets de type *contacts* ;
- les *commands* qui définissent les commandes qui sont lancées par *Shinken* pour vérifier, notifier, etc. ;
- les *time periods*, qui définissent les périodes pendant lesquelles le monitoring et les notifications sont actifs ;
- *notification escalations* : ces objets sont optionnels, ils définissent un processus de notification pour un service. Différents contacts sont notifiés au fur et à mesure de l'aggravation d'un problème.

On retrouve globalement le modèle *Nagios*. *Shinken* apporte toutefois quelques améliorations par rapport à *Nagios*. Il s'agit principalement de définitions de critères de dépendance : *host dependancy*[34] et *services dependancy*[35]. Par ailleurs, la syntaxe des clauses autorise une meilleure maîtrise des questions liées à l'héritage multiple[36] que dans *Nagios*. Enfin, *Shinken* introduit la notion de *packs*, en réalité des modèles intégrés qui prédéfinissent un ensemble d'objets hôtes, services et commandes.

34 https://shinken.readthedocs.org/en/latest/08_configobjects/hostdependency.html

35 https://shinken.readthedocs.org/en/latest/08_configobjects/servicedependency.html

36 https://shinken.readthedocs.org/en/latest/07_advanced/objectinheritance.html

Par exemple, la clause suivante définit un « hôte » qui cible un routeur de marque Cisco. Ses attributs définissent un nom et une adresse IP. Une référence désigne un groupe de contact pour les notifications et la clause *« use »* désigne un modèle défini en *pack* :

```
define host{
        use                     cisco3760
       contact_groups          admins
        host_name               cisco_gateway
        address                 192.168.254.10
        process_perf_data               1
        _ports Gi1/0/1,Gi2/0/3,Gi2/0/1,Fa1/0/23,Gi2/0/2,Gi2/0/4,Gi1/0/2,Fa1/0/1
        }
```

Le *pack cisco3760* définit un modèle d'hôte :

```
define host{
   name           cisco3760
   use            generic-host
   _SNMPCOMMUNITY    $SNMPCOMMUNITYREAD$
   register       0

  _SWITCH_TIMEOUT    60

  _SWITCH_CPU_LOAD_CRIT    90
  _SWITCH_CPU_LOAD_WARN    80

  _SWITCH_MEMORY_USAGE_CRIT    90
  _SWITCH_MEMORY_USAGE_WARN    80

}
```

Modèle auquel on associe une série de modèles de service :

```
define service{
   service_description    InterfaceErrors
   use                    generic-service
   register               0
   host_name              cisco3760
   check_command          check_switch_interface_errors
   notifications_enabled   1
}
define service{
   service_description    InterfaceStatus GW
   use                    generic-service
   register               0
   host_name              cisco3760
   check_command          check_switch_hardware_health3760
}

define service{
   service_description    CPUStatus 3760 GW
   use                    generic-service
   register               0
   host_name              cisco3760
   check_command          check_switch_cpu3760
}

define service{
   service_description    MemoryStatus  3760 GW
   use                    generic-service
   register               0
   host_name              cisco3760
   check_command          check_switch_memory3760
}
# vérification de l'utilisation de bande passante pour les ports spécifiés en argmument
define service{
   service_description    Interface $KEY$ Cisco Gateway
   use                    generic-service
   register               0
```

```
    host_name               cisco3760
    check_command           check_switch_interface_GW!$KEY$
    duplicate_foreach       _ports
 }
```

Chaque service se réfère à une commande *(check_command)* définie par exemple de la façon suivante :

```
 define command {
      command_name     check_switch_interface_GW
      command_line     /usr/bin/python $PLUGINSDIR$/cisco.py -H $HOSTADDRESS$ -C
$_HOSTSNMPCOMMUNITY$ -D $ARG1$
 }
```

Cette factorisation du code de configuration, quoiqu'apparemment laborieuse, permet de réutiliser et de généraliser le code des commandes ou des services.

3.1.2.2. *Plug-ins*

Les *plug-ins* sont simplement les programmes qui effectuent des vérifications et retournent les informations à *Shinken*. Ils sont exécutés par le *daemon Poller*. Comme *Shinken* utilise les mêmes spécifications que *Nagios* pour les *plug-ins*, les *plug-ins Nagios* sont compatibles avec *Shinken* et inversement. Ces programmes sont soit compilés - comme c'est le cas pour beaucoup de sondes *Nagios* écrites en C - soit interprétés lorsqu'il s'agit de langages de script de type Python, Perl ou Shell.

De fait, il est préférable que ces programmes aient de faibles latences, de façon à ne pas encombrer l'ordonnanceur. La question de la performance d'une solution de supervision comme *Shinken* est particulièrement liée à la qualité des *plug-ins*. Ces programmes sont paramétrés dans les définitions d'objets de type *command*.

Ces commandes sont exécutées à intervalles réguliers. C'est le paramètre *check_interval* qui assure cette régulation. L'objectif étant d'obtenir l'état des éléments concernés le plus exact possible, l'intervalle de vérification doit permettre d'arbitrer entre la précision des données récoltées et la performance induite par l'exécution répétée de cette commande. Ces commandes peuvent solliciter des API externes *(VMware Perl SDK*, *Navisphere Client*, *smiclient*, etc.*)*, pourvu que leur sortie *(output)* respecte le formalisme *Nagios*.

3.1.2.3. *Triggers*

Les *triggers* sont de petits éléments de programmes qui peuvent être déclenchés après le changement d'état d'un objet. Écrits en Python, le code des *triggers* permet de modifier l'état ou les données de l'objet intercepté. Les *triggers* sont relativement peu documentés, mais on peut saisir assez rapidement leur usage en consultant le code source du fichier *trigger_functions.py*[37]. Celui-ci regroupe les méthodes qui peuvent être invoquées lors de l'instanciation d'un objet de type *trigger* : on peut rechercher des ob-

37 https://github.com/naparuba/shinken/blob/master/shinken/trigger_functions.py

jets, hôtes ou services (fonctions *get_object get_objects*), récupérer leurs attributs *(output, perfdata, return_code)* et manipuler ces derniers (fonctions *set_value()*, *set_critical()*, etc.). Concrètement, il est donc possible de recalculer des indicateurs à partir de l'état de différents objets. C'est un moyen de calculer des KPI.

À titre d'exemple, le code ci-dessous a été écrit pour cumuler les volumes d'impression des 8 copieurs multifonctions du SI. On cherche à évaluer le volume global d'impression sans que la mise en veille d'un des copieurs multifonctions perturbe le calcul. Chaque copieur multifonctions est donc associé au même *trigger*, dont le code figure ci-dessous :

```
import pickle
lastmaxbigcounter={}
lastmaxbigcounter['pc']=0
#get the last max bigcounter
try:
    output = open( "/usr/local/Shinken/var/bcmax.p", "r" )
except IOError:
    lmbc=lastmaxbigcounter['pc']
else:
    lastmaxbigcounter=pickle.load(output)
    output.close()
    lmbc=lastmaxbigcounter['pc']

names=get_objects('*/Compteur')
bigcounter=0
#calcul compteur total
for name in names:
    if name.state=='OK':
        myperf=all_perf(name,'pagecount')
        bigcounter=myperf.value+bigcounter

#faire persister le compteur si une des imprimantes est éteinte
if bigcounter>lmbc:
    lastmaxbigcounter['pc']=bigcounter
    output = open( '/usr/local/shinken/var/bcmax.p', 'w' )
    pickle.dump(lastmaxbigcounter, output )
    output.close()
else : bigcounter=lmbc
self.perf_data = self.perf_data + ' TotalpageCount='+ str(bigcounter) + ';;;;;'
```

On utilise le module *pickle*[38] pour stocker la dernière valeur connue et on cumule les différents volumes d'impression. On ajoute enfin un nouvel indicateur dans les données de performance nommé *TotalpageCount*.

On peut, sur la base de ce principe, construire des indicateurs complexes de façon programmatique.

3.1.2.4. Règles métier (business rules)

Un des apports intéressants de *Shinken* par rapport à *Nagios* est sans doute la possibilité de mettre en place des *règles métier*. Ces règles permettent d'associer n'importe quel état d'un objet, hôte ou service, à un nouvel état et à un niveau de criticité. D'une certaine façon, cela permet de s'extraire de l'association systématique d'un *hôte* à des *services*.

38 https://docs.python.org/2/library/pickle.html

L'exemple ci-dessous permet d'évaluer si l'infrastructure réseau délocalisée dans les agences est dans un état de fonctionnement « normal ». Chaque agence de GBH est en effet dotée d'un commutateur. On cherche ici à vérifier l'état d'une chaîne de quatre commutateurs :

```
 define service{
      use         generic-service
      host_name   cisco_gateway
      service_description  cisco agences
           check_command               bp_rule!(cisco_gateway)  &  (cisco_ouest)  &  (cisco_est)  &
(cisco_planoise )
      check_interval               1
      retry_interval               5
      max_check_attempts           2
      business_impact    4
 }
```

Le degré de criticité est paramétré par la clause *« business_impact »*, qui peut prendre une valeur de 1 à 5. L'évaluation de la règle *(bp_rule)* peut être assortie d'opérateurs booléens *(et/ou)* ou de sélecteurs *(1of (x,y))* qui permettent de prendre en compte les facultés de résilience de certaines architectures. Ici, il n'existe aucun mécanisme de ce type dans la chaîne de routage, on applique donc une association stricte : tout élément de la chaîne dont l'état n'est pas normal invalide le bon fonctionnement de la chaîne.

3.1.2.5. Les modules

Les modules *Shinken* sont des programmes qui tirent parti du bus logiciel de *Shinken*. Ils sont capables d'intercepter les objets Python représentant un service ou un hôte et de travailler sur ses propriétés. Contrairement aux *triggers*, ces modules sont associés à un des *daemons* de *Shinken* selon la fonction qu'ils mettent en œuvre. Les informations transitent via des files *(queue)* sous forme d'objets et sont interceptés selon leur état par les processus des modules. *Shinken* fournit un jeu de modules type[39] à partir desquels il est relativement aisé d'implémenter un module spécifique.

Par exemple, le module *Livestatus*[40] est chargé d'assurer la possibilité de communiquer les informations de supervision de *Shinken* via une socket TCP. L'API *Livestatus[89]*, est particulièrement utilisée pour transférer les états collectés par un système de supervision vers d'autres programmes comme *Thruk[90]* ou *Splunk[91]*. *Shinken* propose sa propre implémentation[92] de l'API originale écrite par Mathias Kettner[89]. Un module spécifique, de type *broker* est associé à cette fonctionnalité.

Le code source du module, comme tous ceux mis à disposition sur le site http://www.shinken-monitoring.io, présente l'avantage de la disponibilité du code. Le module *Livestatus* s'articule suivant la logique décrite dans l'Illustration 3.5. On en re-

39 Voir les *Dummy modules* sur https://github.com/naparuba/shinken/tree/master/modules qui décrivent le code Python type d'un module.

40 http://www.shinken.io/package/livestatus

trouve les principaux axes dans le fichier *module.py* disponible sur le dépôt *git* du projet[41].

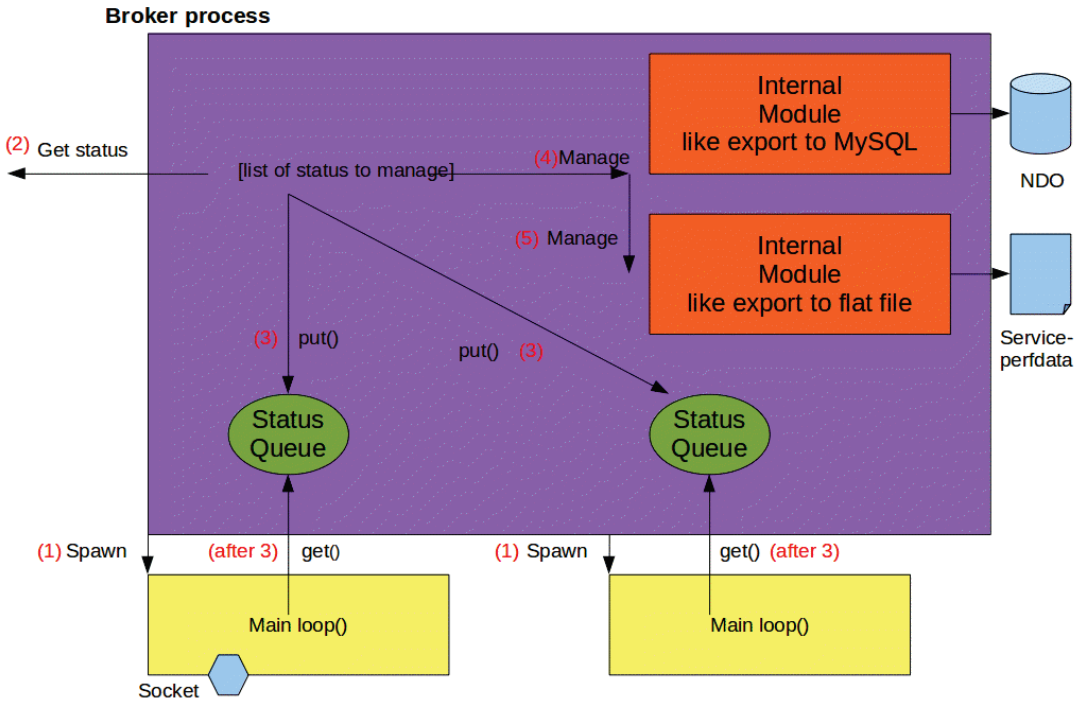


Illustration 3.5 : documentation Shinken - description des actions effectuées dans un module de type broker

Concernant ce module, on distingue :

- la phase d'initialisation du *broker Livestatus* : le module récupère des éléments de paramétrage dans le fichier de configuration principal et effectue des vérifications avant son démarrage à proprement parler :

```
[...]
class LiveStatus_broker(BaseModule, Daemon):

    def __init__(self, modconf):
        BaseModule.__init__(self, modconf)
        # We can be in a scheduler. If so, we keep a link to it to speed up regenerator phase
        self.scheduler = None
        self.plugins = []
        self.use_threads = (getattr(modconf, 'use_threads', '0') == '1')
        self.host = getattr(modconf, 'host', '127.0.0.1')
        if self.host == '*':
            self.host = '0.0.0.0'
        self.port = getattr(modconf, 'port', None)
        self.socket = getattr(modconf, 'socket', None)
        if self.port == 'none':
```

41 Module *Livestatus* : https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/module.py

```
            self.port = None
        if self.port:
            self.port = int(self.port)
        if self.socket == 'none':
            self.socket = None
        self.allowed_hosts = getattr(modconf, 'allowed_hosts', '')
        ips = [ip.strip() for ip in self.allowed_hosts.split(',') if ip]
 [...]
```

- puis la préparation de l'instanciation : on définit une file de communication avec le module *broker* :

```
 def init(self):
        logger.info("[Livestatus Broker] Init of the Livestatus '%s'" % self.name)
        self.prepare_pnp_path()
        m = MacroResolver()
       m.output_macros   =   ['HOSTOUTPUT',   'HOSTPERFDATA',   'HOSTACKAUTHOR',   'HOSTACKCOMMENT',
'SERVICEOUTPUT', 'SERVICEPERFDATA', 'SERVICEACKAUTHOR', 'SERVICEACKCOMMENT']
        self.rg.load_external_queue(self.from_q)
```

- le module *Livestatus* charge ensuite ses propres dépendances. Il s'agit ici d'autres modules qui correspondent au type de base de données qui a été indiqué en configuration : les informations stockées par ce module peuvent utiliser une base *Sqlite* ou *MongoDB* :

```
    def main(self):
        self.set_proctitle(self.name)

        self.log = logger
        self.log.load_obj(self)
        # Daemon like init
        self.debug_output = []
        self.modulesdir = modulesctx.get_modulesdir()
        self.modules_manager = ModulesManager('livestatus', self.modulesdir, [])
        self.modules_manager.set_modules(self.modules)
        # We can now output some previously silenced debug output
        self.do_load_modules()
        for inst in self.modules_manager.instances:
            if inst.properties["type"].startswith('logstore'):
                f = getattr(inst, 'load', None)
                if f and callable(f):
                    f(self)
                break
        for s in self.debug_output:
            logger.debug("[Livestatus Broker] %s" % s)
        del self.debug_output
        self.add_compatibility_sqlite_module()
        self.datamgr = datamgr
        datamgr.load(self.rg)
        self.query_cache = LiveStatusQueryCache()
        if not self.use_query_cache:
            self.query_cache.disable()
        self.rg.register_cache(self.query_cache)

        try:
     [...]
            self.do_main()
        except Exception, exp:
            msg = Message(id=0, type='ICrash', data={
                'name': self.get_name(),
                'exception': exp,
                'trace': traceback.format_exc()
            })
        self.from_q.put(msg)
        # wait 2 sec so we know that the broker got our message, and die
        time.sleep(2)
        # (try to) clean before exit
        self.do_stop()
        raise
```

- l'action principale est lancée dans la fonction *do_main* par la boucle précédente. On y distingue un *thread* qui travaille sur les requêtes *Livestatus (lql_thread)* et un autre qui stocke les données en provenance du *broker Shinken (data_thread)* :

```
    def do_main(self):
        [...]

        # Open the logging database
        self.db = self.modules_manager.instances[0]
        self.db.open()

        # We ill protect the operations on
        # the non read+write with a lock and
        # 2 int
        self.global_lock = threading.RLock()
        self.nb_readers = 0
        self.nb_writers = 0

        self.data_thread = None
 [...]

        self.load_plugins()

        if self.use_threads:
            # Launch the data thread"
            logger.info("[Livestatus Broker] Starting Livestatus application")
            self.data_thread = threading.Thread(None, self.manage_brok_thread, 'datathread')
            self.data_thread.start()
            self.lql_thread = threading.Thread(None, self.manage_lql_thread, 'lqlthread')
            self.lql_thread.start()
            self.data_thread.join()
            self.lql_thread.join()
        else:
            self.manage_lql_thread()
```

- la fonction *« manage_brok_thread »* intercepte les messages qui sont mis à disposition dans la file et les traite, ici en les renvoyant à un autre module de type *broker* via la fonction *« manage_brok »*, qui permet de stocker les informations dans une base de données ;

```
    def manage_brok_thread(self):
        logger.info("[Livestatus Broker] Data thread started")
        while True:
            l = self.to_q.get()
            for b in l:
                # Un-serialize the brok data
                b.prepare()
                # For updating, we cannot do it while
                # answer queries, so wait for no readers
                self.wait_for_no_readers()
                try:
                    logger.debug("[Livestatus Broker] Got data lock, manage brok")
                    self.rg.manage_brok(b)
                    for mod in self.modules_manager.get_internal_instances():
                        try:
                            mod.manage_brok(b)
                        except Exception, exp:
                            logger.debug("[Livestatus Broker] %s" % str(exp.__dict__))
                            logger.warning("[%s] The mod %s raise an exception: %s, I'm tagging it
to restart later" % (self.name, mod.get_name(), str(exp)))
                            logger.debug("[%s] Exception type: %s" % (self.name, type(exp)))
                            logger.debug("Back trace of this kill: %s" % (traceback.format_exc()))
                            self.modules_manager.set_to_restart(mod)
                except Exception, exp:
                    msg = Message(id=0, type='ICrash', data={
```

```
                        'name': self.get_name(),
                        'exception': exp,
                        'trace': traceback.format_exc()
                    })
                    self.from_q.put(msg)
                    # wait 2 sec so we know that the broker got our message, and die
                    time.sleep(2)
                    # No need to raise here, we are in a thread, exit!
                    os._exit(2)
```

Ce module confère à *Shinken* la possibilité d'être interrogé par une application distante en utilisant la syntaxe de requêtes *LiveStatus*. Par exemple, en Python, on obtient très simplement le statut des hôtes ou services avec le code suivant :

```
import pprint
import socket
s = socket.socket()
target = ('172.16.1.14',50000)
commandHosts="GET status\n"
s.send(commandHosts)
answer = s.recv(1000)
table = [ line.split(';') for line in answer.split('\n')[:-1] ]

print answer
```

En appliquant ce principe, les modules permettent d'interfacer *Shinken* avec de nombreuses fonctionnalités ou des programmes externes. On en trouve une liste exhaustive sur la documentation officielle de *Shinken*[42].

L'implémentation *Shinken* utilise des méthodes s'apparentant à celles décrites par le *Data Driven Architecture[93]*: les processus s'échangent les données - en fait des objets Python, qui peuvent retranscrire l'état d'un hôte ou d'un service et le transforment au gré du processus de supervision. Cela consiste *à ne plus coder « en dur » les opérations que l'on fait sur les propriétés d'un objet, mais à lire ces opérations dans un tableau. Ce dernier sera un dictionnaire dont les clés sont les noms des attributs sur lesquels nous allons opérer, et les valeurs sont les opérations à effectuer, typiquement une fonction[94].*

Ces modules sont un liant indispensable à l'enrichissement de la mécanique de supervision. Un certain nombre d'entre eux ont été exploités dans le cadre de ce projet de supervision. Il s'agit principalement de l'interface web *(webui[95])*, du module *WS-arbiter[96]* qui facilite l'acquisition de données passives par HTTP, du module *MongoDB[97]*, qui permet de stocker les informations dans une base de données éponyme et du module *Canopsis[98]*, qui permet de transférer les données de supervision à un hyperviseur *Canopsis* distant.

3.1.3. Présentation des données et interfaces graphiques

La question de la représentation graphique des données de supervision est le fait de nombreux produits qui ont été évoqués au paragraphe 2.3.2. *Shinken* propose sous forme de modules l'inclusion d'applications comme *Thruk*, *Nagvis*, PNP ou *Graphite*. Chacune

42 http://www.shinken-monitoring.org/wiki/official/shinkenaddons-addons?s[]=modules

de ces solutions a été testée puis abandonnée, à l'exception de *Graphite*. En effet, ces outils sont spécialisés dans la représentation graphique de métriques de performance. Or, ces derniers offrent finalement peu de possibilités interactives de manipulation des données de performance. *Graphite[99]* présente toutefois des capacités d'agrégation et de calcul interactives relativement innovantes.

L'abandon des outils « classiques » de représentation des données est lié au constat qu'au fur et à mesure de la mise en place d'indicateurs de performance, le fait de disposer de milliers de ceux-ci ne pouvait favoriser d'emblée la lisibilité du système d'information. Aussi, l'intention de représenter une donnée de performance de façon systématique a semblé dépourvue de sens. Au contraire, les solutions qui proposent une composition graphique interactive permettent de sélectionner, d'agréger et de recalculer des données de performance en y associant une représentation graphique dynamique.

3.1.3.1. L'interface Shinken

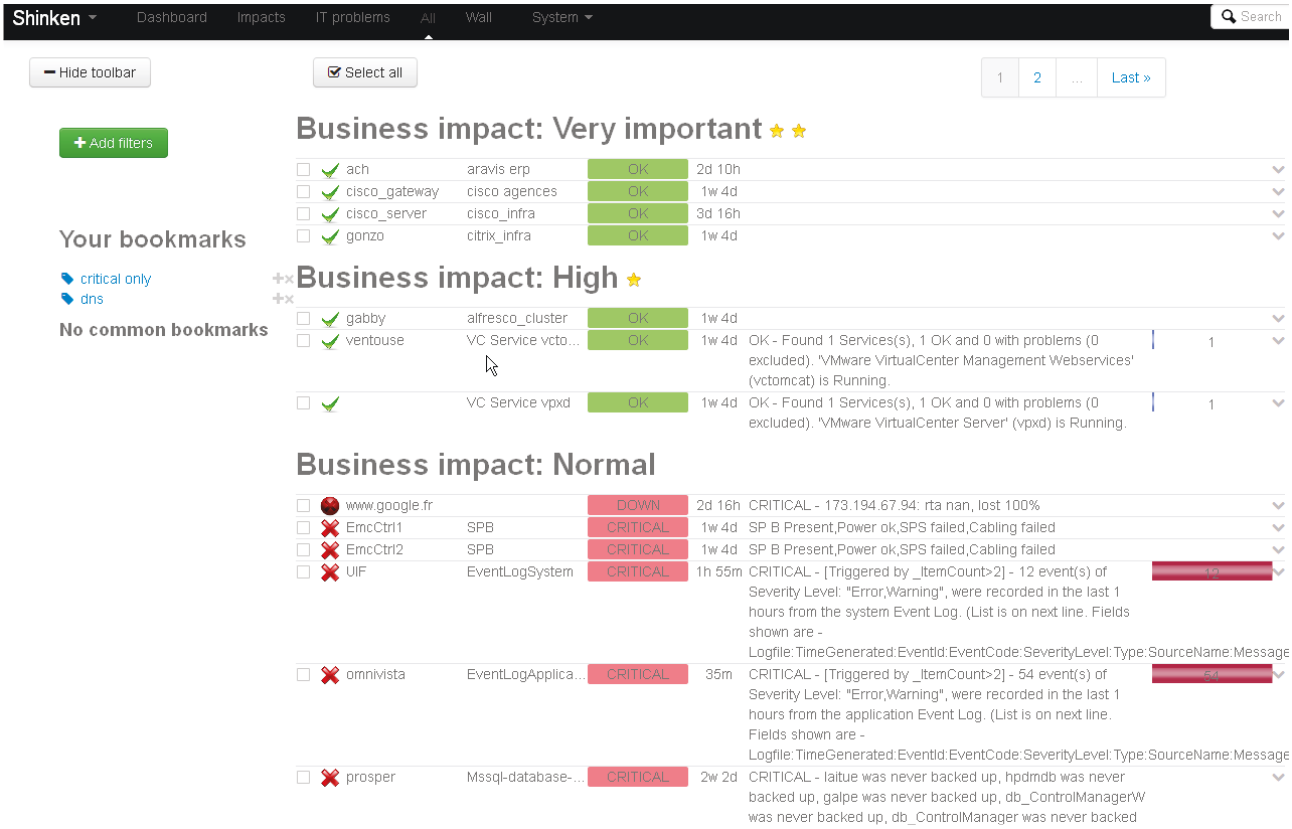


Illustration 3.6 : vue de l'interface web Shinken

Shinken propose sa propre interface graphique qui repose sur le module *webui[95]*. Relativement sobre, elle distingue les problèmes techniques (IT) des problèmes métier *(business)*. On retrouve là la distinction entre les vérifications techniques des sondes -

qui concernent davantage les techniciens de l'infrastructure - et les *business rules* évoquées plus haut, qui s'attachent à restituer l'état des éléments en termes stratégiques et métier. Le développement de *Shinken* n'est toutefois clairement pas orienté vers la représentation des données de performance. Ces questions sont déléguées à des applications tierces, comme *Nagvis* ou *Thruk*, qui communiquent avec *Shinken* à l'aide du protocole *Livestatus*, ou encore *Canopsis*, qui communique avec *Shinken* via l'API *kombu[100]* et le protocole AMQP[101].

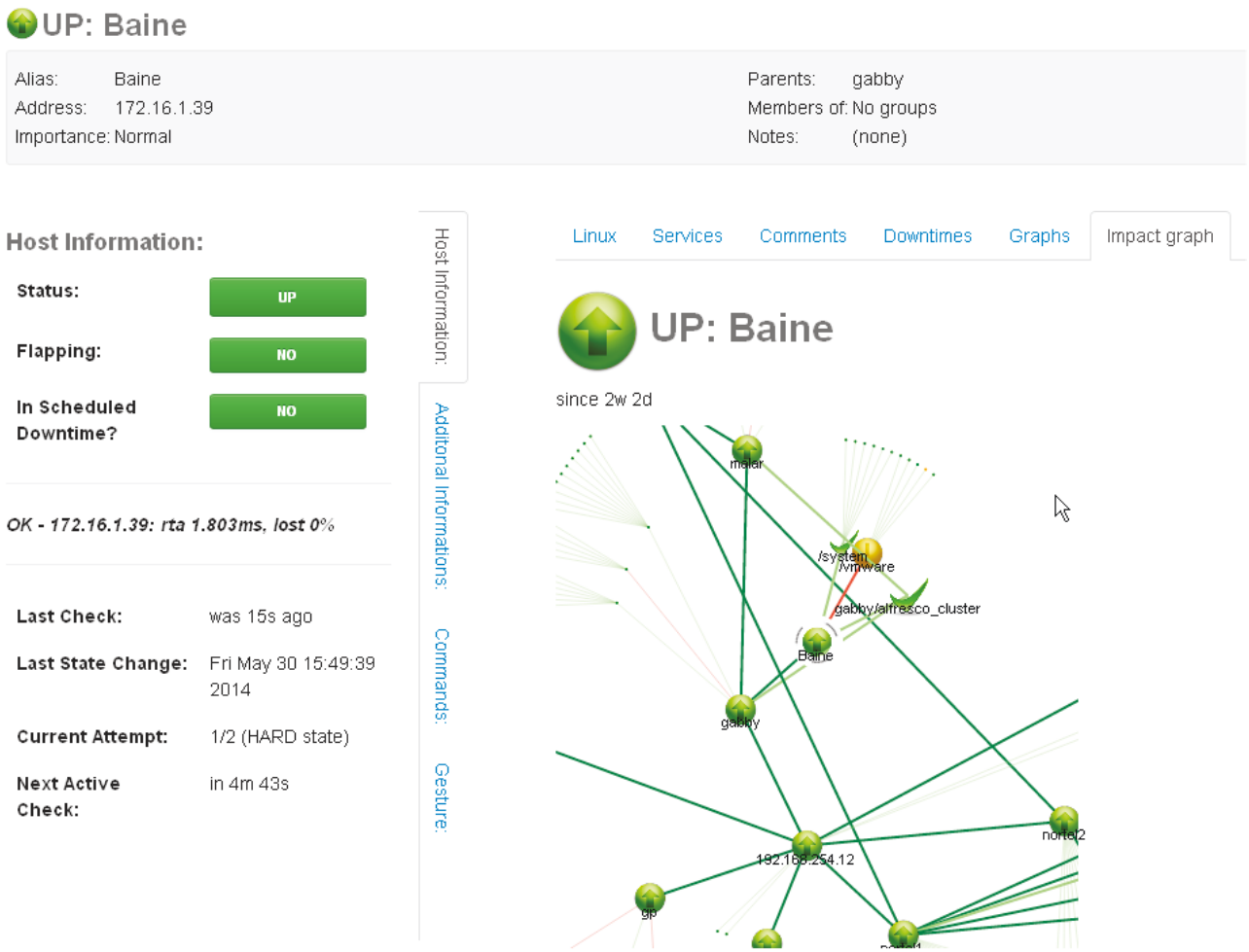


Illustration 3.7 : représentation des dépendances dans l'interface webui

L'interface graphique de *Shinken* se veut efficace et simple, sans doute dans l'intention d'éviter de noyer l'utilisateur sous un flot d'informations. Dans les cas de dépendances multiples entre hôtes et services, la difficulté est d'identifier la cause initiale du problème, plutôt que les conséquences. La visualisation d'alertes massives ne permet en effet pas nécessairement de faire la distinction entre l'origine et les conséquences d'un problème. C'est à ce niveau que la qualité de l'interface graphique tient un rôle impor-

tant pour l'opérateur. Mais si cette interface démontre des qualités opératoires évidentes, elle reste toutefois peu lisible pour les novices. Or, cela ne correspond pas à l'intention initiale du projet, qui vise justement un travail sur la représentation du SI et son sens pour les utilisateurs. Il est donc rapidement apparu que l'interface *Shinken* ne suffisait pas.

Quelques essais avec *Thruk[90]* et *Nagvis[102]*, ont montré d'intéressantes facultés de représentation, là encore en favorisant très largement une optique technique du SI. La solution *Canopsis* - quoique plus lourde à mettre en œuvre - restait la seule offrant d'une part la capacité de communiquer avec *Shinken* et d'autre part celle d'associer des indicateurs complexes avec des représentations graphiques synthétiques.

3.2. Canopsis

Canopsis est une solution open-source développée par la société Capensis. Sur le site de celle-ci[43], on trouve la définition suivante : *Canopsis est un environnement d'hypervision. Positionné au sein de l'infrastructure au-dessus des solutions de supervision génériques ou spécialisées, ce puissant outil permet de corréler et d'agréger un grand nombre d'événements provenant de sources diverses. La manipulation pertinente de ceux-ci permet de réconcilier vision technique et vision métier du système d'information[44].* On note, comme dans le cas de *Shinken*, que ce projet met en avant la dualité des visions technique et métier.

3.2.1. Fonctionnement général

Globalement, *Canopsis* agit de la même façon qu'un ETL dans une chaîne décisionnelle : il collecte les données via ses connecteurs AMQP. Des moteurs de traitement *(engines)* transforment les données extraites de différentes sources (SNMP, *Syslog*, *Shinken*) à la façon d'un *Operational Data Store*[45]. Les données transformées sont stockées dans une base *MongoDB*, qui fait office de *dataware house*. On y accède via une interface web ou des outils externes de présentation des données. L'Illustration 3.8 synthétise ces interactions. S'y distinguent une logique de collecte qui repose sur des connecteurs, une mé-

43 Site de la société Capensis : http://www.capensis.fr/

44 Voir plus spécifiquement : http://www.capensis.fr/solutions/hypervision/

45 « Un ODS (ou *Operational data store*) est une base de données conçue pour centraliser les données issues de sources hétérogènes afin de faciliter les opérations d'analyse et de *reporting*. L'intégration de ces données implique souvent une purge des informations redondantes. Un ODS est généralement destiné à contenir des données de niveau fin comme un prix ou le montant d'une vente, en opposition aux données agrégées tel que le montant total des ventes. Les données agrégées sont stockées dans un entrepôt de données *(data warehouse)*. »[103]

canique interne de stockage qui s'appuie sur le bus AMQP et une base *MongoDB*. Enfin, une logique de présentation repose sur des *frameworks* graphiques et met également en œuvre des facilités d'exportation.

Illustration 3.8 : architecture de Canopsis extrait de http://wiki.monitoring-fr.org/canopsis/canopsis-work

Canopsis dénote une certaine complexité, tant il sollicite des couches applicatives variées pour mettre en œuvre son procédé d'*hypervision* : un serveur *Redis[104]* se charge de collecter les données, les garde en mémoire et les écrit à intervalles réguliers dans la base *MongoDB*. Le « bus » AMQP est tenu par un serveur *Celeryd[105]* qui agit comme un ordonnanceur de tâches. Ceci permet à *Canopsis* de gérer des tâche synchrones - - comme la réception des nouveaux messages - et des tâches asynchrones - comme les calculs de SLA et l'archivage des données de performance.

Ces tâches sont effectuées par des moteurs *(engines)* qui s'articulent donc en deux catégories : les moteurs synchrones traitent les files d'événements provenant des connec-

teurs et les moteurs asynchrones opèrent sur les données stockées dans la base *Redis* ou *MongoDB* pour effectuer, par exemple, des opérations d'agrégation.

Shinken est également capable d'effectuer des opérations sur les données de supervision, par exemple avec les *triggers*, mais n'effectue pas d'actions asynchrones. D'une certaine façon - et c'est là leur limite - il faut rester extrêmement vigilant à l'impact des calculs programmés par l'intermédiaire de *triggers* sur la latence globale de *Shinken*. L'objectif étant de pouvoir se rapprocher au mieux de l'état du système à un instant donné, le calcul d'indicateurs complexes peut être un facteur de latence. L'intérêt de dissocier la collecte de données en temps réel et les logiques de transformation et de présentation des données reflète la distinction entre l'instance de collecte de données de production - faite sur l'instant - et les logiques de transformation (agrégation, composition d'indicateurs et de KPI, analyses statistiques), qui s'effectuent à posteriori. Dans l'Illustration 3.9, on distingue le chaînage des moteurs - qui traitent les événements et alertes – des actions qui les transforment en données de performance.

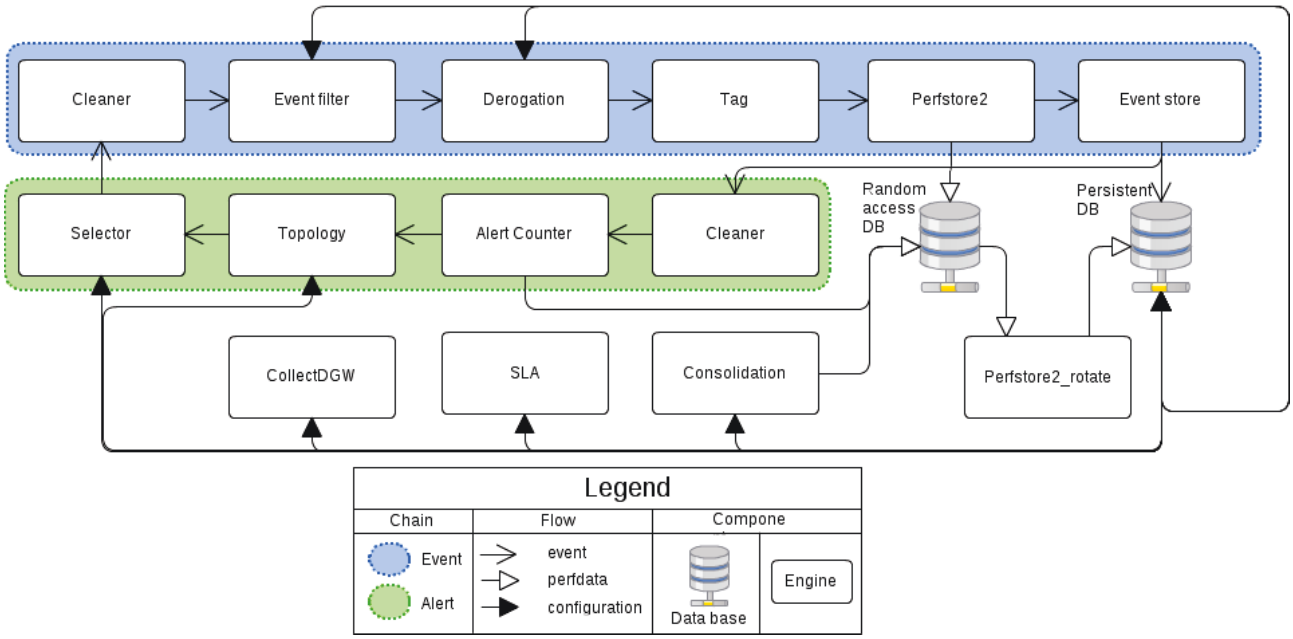


Illustration 3.9 : Canopsis Engines

Les messages traités par *Canopsis* doivent respecter une structuration simple formalisée de la façon suivante[46] :

```
{
  '_id':              (reserved) MongoDB
  'event_id':         (reserved) Original in alerts exchange
  'connector':        Connector type (gelf, nagios, snmp, ...),
  'connector_name':   Connector name (nagios1, nagios2 ...),
  'event_type':       Event type (check, log, trap, ...),
  'source_type':      Source type ('component' or 'resource'),
```

46 https://github.com/capensis/canopsis/wiki/Event-specification

```
    'component':         Component name,
    'resource':          Ressource name,
    'timestamp':         UNIX seconds timestamp (UTC),
    'state':             State (0 (Ok), 1 (Warning), 2 (Critical), 3 (Unknown)),
    'state_type':        State type (0 (Soft), 1 (Hard)),
    'scheduled':         (optional) True if this is a scheduled event
    'last_state_change': (reserved) Last timestamp after state change,
    'previous_state':    (reserved) Previous state (after change),
    'output':            Event message,
    'long_output':       Event long message,
    'tags':              Event Tags (default: []),
    'display_name':      The name to display (customization purpose)
  }
```

En s'appuyant sur la logique d'enregistrements de type « clé-valeur » caractéristique des bases dites *NoSQL*, *Canopsis* mise sur la performance du système *MongoDB[106]*. Des enregistrements peuvent y être stockés suivant un encodage BSON *(Binary JSON)* tout en intégrant des structures de données complexes. L'intérêt de cette base *NoSQL*, outre ses performances, est qu'elle est dépourvue de schéma de données ou de relations : tout enregistrement représentant un objet *Canopsis* doit uniquement être pourvu d'un identifiant. De fait, les insertions de données sont simplifiées par rapport à une base relationnelle. Les différentes données de *Canopsis* s'organisent en *collections* qui structurent les événements et les données de performance.

Les données de performance sont archivées dans les collections *perfdata_bin* suivant les spécifications *GridFS[107]*. Les méta-données et les données de performance récentes sont stockées sans compression dans la collection *perfdata*. L'exemple ci-dessous illustre un requêtage simple sur la collection *perfdata* :

```
  > show collections
  beaker
  binaries.chunks
  entities
  events
  events_log
  object
  perfdata2
  perfdata2_bin.chunks
  perfdata2_bin.files
  perfdata2_daily
  system.indexes
  > db.perfdata2.find()
  { "_id" : "9f7e426a517c16893cd3ca0671fbfcb6", "co" : "shi-komputer", "lts" : 1400321484, "lv" :
237224,  "me"  :  "user",  "mi"  :  0,  "r"  :  0,  "re"  :  "cpu-0",  "t"  :  "DERIVE",  "tg"  :
[ "collectd2event", "check", "resource", "shifuck-komputer", "cpu-0" ] }
  [...]
```

Un des autres avantages du stockage *MongoDB* réside dans la grande variété d'API qui permettent d'accéder à ses données. Dans le cas de *Canopsis*, Python et Javascript sont les principaux langages sollicités. Les interactions entre *MongoDB* et Javascript permettent notamment la mise en place de graphes dans l'interface web.

Le serveur web repose sur *Gunicorn[108]*, un serveur HTTP dédié aux applications

web écrites en Python et qui supporte nativement le WSGI[109]. *Canopsis* met en œuvre une API REST *(Representationel State Transfert)* qui est une architecture pour les médias distribués, décrite par Roy Thomas Fielding dans sa thèse *Architectural Styles and the Design of Network-based Software Architectures[110]*. Dans *Canopsis*, le service web REST permet par exemple d'accéder aux éléments de base *MongoDB* en suivant la syntaxe suivante :

http://serveurcanopsis/rest/:namespace/:ctype/:_id

dans laquelle :

- :*namespace* représente la collection *MongoDB* ;
- *:ctype* représente le type d'objet (compte utilisateur, groupe, vue, événement, etc.).

Les différents types de requêtes HTTP (*Get, Put, Delete, Post*) du service REST permettent de manipuler tous ces enregistrements. Les requêtes retournent, le cas échéant, des objets encodés au format JSON. Cette facilité permet par exemple d'exploiter les métriques par l'intermédiaire de programmes JavaScript capables de manipuler le JSON.

Le service web REST est très générique, mais l'API web expose un certain nombre d'autres services accessibles selon des modalités similaires. Parmi eux, on dénombre les services a*uth* (authentification), e*vent* (événements), f*ile* (fichiers), p*erfstore* (métriques de performances), *reporting* (export de vues), *right* (droits d'accès), v*iew* (vues graphiques), etc. La syntaxe de chacun de ces services web est décrite plus précisément dans le wiki *Canopsis*[47].

D'un point de vue très général, chaque événement reçu par *Canopsis* est produit par un connecteur. Le type de l'événement est lié à sa nature : une *trap* SNMP, une information en provenance d'un serveur *Syslog* distant, une alerte *Shinken*, etc. Cet événement est lié à un composant *(component)* et à une ressource *(resource)*. Dans le cas d'un événement compatible avec la syntaxe *Nagios*, le composant désignerait un hôte et les ressources ses services.

Concernant les moteurs de traitement *(engines)*, on distingue notamment :

- le moteur *consolidation* qui permet d'agréger diverses données de performance ;
- le moteur de *SLA* : permet de définir des règles de calculs vérifiant la disponibilité des infrastructures supervisées ;
- le moteur *selectors* : les sélecteurs permettent de calculer l'état général d'un lot d'événements en appliquant des algorithmes simples. Par exemple, en appliquant

47 https://github.com/capensis/canopsis/wiki/API-Web

l'algorithme de type *« worst state »* : l'état du sélecteur correspond à l'état le plus critique des composants qu'il englobe. On peut, de cette façon, restituer des liens de dépendance logiques entre les composants supervisés ;

- le moteur de *dérogation* permet de modifier l'état d'un événement en lui appliquant une règle spécifique : cela permet, par exemple, de transformer un événement de supervision en lui affectant la transformation systématique de ses états *critical* en état *OK* ;
- le moteur *topology* : très proche de la notion de *business rules* que l'on trouve dans *Shinken*, ce moteur permet de vérifier des règles simples entre les composants, ressources ou sélecteurs. Il s'agit en fait de créer un chaînage logique.

La typologie de *Canopsis* se veut très large, ou du moins capable d'intégrer un grand nombre de « messages » en provenance de divers outils de supervision : alertes de type *Nagios*, *Syslog* distants ou *trap* SNMP. L'intérêt majeur des transformations proposées par *Canopsis* réside dans leur capacité à exploiter ces différentes formes de messages pour constituer un KPI : contrairement aux outils de rendu graphique traditionnels, *Canopsis* offre la possibilité d'exploiter des événements et pas uniquement des métriques. De cette façon, le calcul d'un SLA pourra représenter le taux d'état *UP* d'un hôte sur une période donnée.

La documentation de *Canopsis* n'est néanmoins pas très développée. On trouve quelques indications pertinentes sur le wiki du dépôt *Github*, mais les éléments les plus intéressants se trouvent probablement dans le code source. La dernière version dite « stable » est dénommée Ficus. Elle date de janvier 2013. La version de la branche *git «develop »* est difficilement exploitable en production du fait de corrections fréquentes. Elle nécessite néanmoins une veille active sur le dépôt *Github* du projet.

3.2.2. Connecteur et messages *Canopsis*

Concernant les messages, les contraintes sont de deux ordres.

1. Une contrainte fonctionnelle : chaque message doit avoir un type précis - snmp, log ou alerte par exemple. Là encore, les indications sont minimalistes, si bien que le bon positionnement des attributs des messages reste dépendant d'expérimentations préalables. Chaque message est identifié par une clé de routage qui est composée des différentes caractéristiques de l'événement. Ainsi, le connecteur *Canopsis* pour *Shinken* classe les événements à exporter selon leur nature.

2. Une contrainte technique : les messages doivent être envoyés sur le bus AMQP du

serveur *Canopsis*, en fait vers une instance d'un serveur *Redis*.

Le code du connecteur se trouve dans un module qui est associé au *broker Shinken* :

- les messages *Shinken* suivent le formalisme décrit dans les commentaires de la fonction *create_message* du fichier source *canopsis.py*[48] :

```
def create_message(self, source_type, event_type, b):
        """
event_type should be one of the following:
- check
- ack
- notification
- downtime

source_type should be one of the following:
- component => host
- resource => service

message format (check):

H S field desc
x 'connector' Connector type (gelf, nagios, snmp, ...)
x 'connector_name': Connector name (nagios1, nagios2 ...)
x 'event_type' Event type (check, log, trap, ...)
x 'source_type' Source type (component or resource)
x 'component' Component name
x 'resource' Resource name
x 'timestamp' UNIX seconds timestamp
x 'state' State (0 (Ok), 1 (Warning), 2 (Critical), 3 (Unknown))
x 'state_type' State type (O (Soft), 1 (Hard))
x 'output' Event message
x 'long_output' Event long message
x 'perfdata' nagios plugin perfdata raw (for the moment)
x 'check_type'
x 'current_attempt'
x 'max_attempts'
x 'execution_time'
x 'latency'
x 'command_name'
'address'
"""
```

- le module compose un message suivant les spécifications *Canopsis* dans la même fonction *create_message* :

```
 commonmessage = {
            'connector': u'shinken',
            'connector_name': unicode(self.identifier),
            'event_type': event_type,
            'source_type': source_type,
            'component': b.data['host_name'],
            'timestamp': b.data['last_chk'],
            'state': b.data['state_id'],
            'state_type': b.data['state_type_id'],
            'output': b.data['output'],
            'long_output': b.data['long_output'],
            'perf_data': b.data['perf_data'],
            'check_type': b.data['check_type'],
            'current_attempt': b.data['attempt'],
            'execution_time': b.data['execution_time'],
            'latency': b.data['latency'],
            'address': self.host_addresses[b.data['host_name']]
        }
```

48 https://github.com/shinken-monitoring/mod-canopsis/blob/master/module/module.py

- la fonction *postmessage* reconstitue une clé de routage qui permet d'indiquer la nature du message à *Canopsis* et de distinguer les hôtes (type *component*) des services (type *resources*) :

```
    def postmessage(self, message, retry=False):

        # process enqueud events if possible
        self.pop_events()

        key = "%s.%s.%s.%s.%s" % (
            message["connector"],
            message["connector_name"],
            message["event_type"],
            message["source_type"],
            message["component"]
        )

        if message["source_type"] == "resource":
            key = "%s.%s" % (
                    key,
                    message["resource"]
                )
[...]
```

- puis il transmet la clé à *Canopsis* en sollicitant l'API AMQP via le module Python *kombu* :

```
if self.connected():
            logger.debug("[Canopsis] using routing key %s" % key)
            logger.debug("[Canopsis] sending %s" % str(message))
            try:
                self.producer.revive(self.channel)
               self.producer.publish(body=message, compression=None, routing_key=key,
                                     exchange=self.exchange_name)
                return True
```

Shinken ne transmet toutefois que les événements de type *'check'*. Il est tout à fait envisageable, suivant le même synopsis, de transformer d'autres types d'événements pour les intégrer à *Canopsis*. Cette technique est mise en œuvre pour « pousser » les événements autres que les *checks* vers *Canopsis*.

3.2.3. Rendu graphique

L'interface web de *Canopsis* permet de :

- créer des indicateurs composés en lien avec les *engines* évoqués plus haut ;
- créer des représentations graphiques des différents indicateurs récoltés ou construits à posteriori à l'aide de ces *engines*. *Canopsis* propose dans ce but une série de *widgets* capables de restituer visuellement l'information ;
- organiser des groupes de *widgets* en « vues » et leur associer des droits d'accès.

Illustration 3.10 : widgets Canopsis

Ces *widgets* permettent de visualiser l'état d'un composant ou des métriques qui y sont associées. Ils s'appuient sur des *frameworks* Javascript comme *ExtJS[111]* pour l'interaction graphique et les boîtes de dialogue, *Jquery[112]* ou *Flot[113]* pour le tracé des graphes.

L'interface graphique s'organise en « vues », chacune d'entre elles pouvant être assortie de droits d'accès spécifiques. Chaque vue est construite par des interactions directes avec les boîtes de dialogue en mode *web*. Une approche programmatique est envisageable en exploitant les API web décrites plus haut, mais les *widgets Canopsis* offrent des possibilités de représentation et de *reporting* suffisamment précises.

Parmi les fonctionnalités intéressantes, on peut citer :

- la gestion des ACL[49], qui permet de limiter les accès des utilisateurs à certaines vues ;
- l'export programmé de vues, qui permet d'envoyer l'état d'une vue sous forme de fichier au format .pdf via SMTP.

Les vues sont classées dans une arborescence thématique établie par le service informatique de GBH. C'est ce que l'on distingue dans l'Illustration 3.11.

49 *Access Control List*

Illustration 3.11 : panneau du gestionnaire de vues Canopsis

3.2.4. L'association Shinken - Canopsis : synthèse

L'association des deux plates-formes, *Shinken* et *Canopsis* permet de dissocier :

- la mise en œuvre des tests de vérification et la récolte des données ;
- la constitution de KPI ;
- la présentation graphique des résultats.

Les deux solutions présentent certaines fonctionnalités très similaires, comme par exemple la constitution de KPI, la capacité de dissocier la partie opératoire où l'on récolte des données brutes de la partie de présentation des données. Celle-ci est semblable

à celle que l'on met en œuvre dans une chaîne décisionnelle. *Shinken* s'attache aux aspects les plus opérationnels de la supervision : son interface graphique s'adresse aux techniciens du SI. De fait, ses capacités de restitution, d'historisation et d'analyse sont contraintes par un impératif de performance : les latences entre le lancement des tests et la restitution du résultat doivent demeurer faibles pour permettre d'alerter des dysfonctionnements en temps réel. *Shinken* travaille sur des données volatiles. De l'autre côté, *Canopsis* met en œuvre des mécanismes désynchronisés, plus aptes à permettre des analyses à posteriori, à corréler différents événements, à les historiser et a en fournir un rendu graphique dont la lisibilité n'est pas limitée aux seuls techniciens.

Les modèles mis en œuvre, quoique peu structurés - 'ressources/composants' sur *Canopsis* et 'hôte/service' sur *Shinken* - offrent en contrepartie une souplesse d'adaptation remarquable. De fait, il est possible d'y intégrer différents types de tests techniques ou applicatifs, pour peu que l'on soit en mesure de mettre en œuvre les automatismes nécessaires.

4. Modalités de mise en œuvre - adaptations au SI de GBH

4.1. Organisation du projet

4.1.1. Moyens de mise en œuvre et mise en place du processus

La mise en place de ce travail s'est étalée sur environ une année. Cette période n'a pas été exclusivement consacrée au projet. Nous avons pris en charge la mise en œuvre des plates-formes ainsi que l'écriture des programmes, notamment des sondes, des *triggers* ou des diverses corrections et adaptations. Notre collègue Renaud Baudiquey a particulièrement contribué au paramétrage de l'infrastructure GBH, à la mise en place des vues *Canopsis* et à l'élaboration de certains indicateurs complexes. Les produits *Shinken* et *Canopsis* avaient fait l'objet d'évaluations préalables, celles-là mêmes qui ont conduit à arrêter un choix définitif. Ces évaluations avaient notamment permis de mettre en place et de systématiser le paramétrage SNMP des composants physiques et logiques du SI.

L'objectif général du projet n'était pas de fournir une solution clés en main, mais plutôt de mettre en place des outils évolutifs, capables de nous aider à superviser et à rendre compte du SI. L'Illustration 4.1 restitue les principales phases de la mise en œuvre du projet. Ainsi qu'on le constate, les cycles d'évaluation et de collecte sont relativement importants. Ils sont liés aux corrections de code et à l'écriture de certains *plug-ins*.

Trois machines virtuelles ont été mises en œuvre : une pour *Shinken* sous un système Linux *Debian 7*, une autre pour *Canopsis* sous le même système d'exploitation et une dernière dédiée à la mise en place des tests comportementaux, nécessitant une interface graphique avec le système d'exploitation Windows Seven.

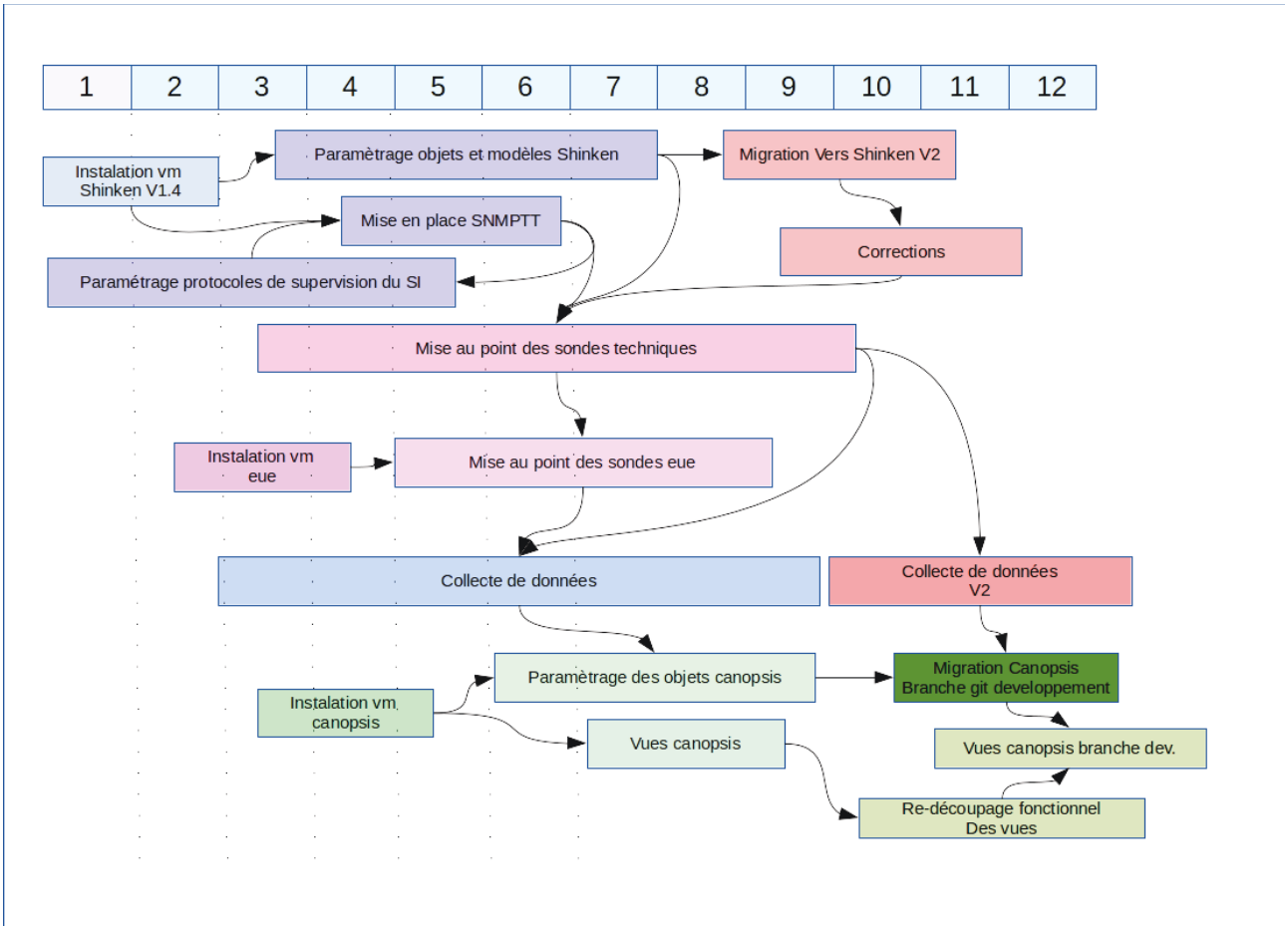


Illustration 4.1 : enchaînement des phases de mise en œuvre du projet de supervision

La calibration des ressources des deux systèmes Linux a posé davantage de problèmes sur le serveur *Shinken*. Une des principales difficultés a consisté à préserver une latence acceptable dans l'exécution des différentes sondes. Outre les questions d'optimisation du code de ces *plug-ins*, il a également fallu tenir compte des évolutions des différentes API sollicitées par ces derniers. Par exemple, l'API *Vsphere*, qui permet de requêter les instances VMware *ESXi* a été la cause d'incidents particulièrement impactants : une mauvaise utilisation des sessions d'identification dans les commandes de vérification a saturé la base de données du *VCenter*, empêchant ainsi son fonctionnement. Par effet de cascade, l'instance *Shinken* a vu ses temps de latence se dégrader au point de devenir quasiment inopérante. Ce cas d'espèce montre bien la fragilité de la solution et la nécessité d'en maîtriser les divers tenants et aboutissants. L'installation de la solution ne pose pas de véritable problème ; les documentations sont relativement précises à ce niveau. Par contre, l'étalement du projet dans le temps s'explique par plusieurs raisons :

- le fait que ce type de projet a traditionnellement un intérêt qui se limite à l'administration système et réseau. En tout état de cause, sa valorisation auprès des autres services dépend fortement des résultats qu'il produit et de leur lisibilité.

L'idée de valoriser une vision stratégique et métier, si elle participe à s'extraire d'une vision techniciste du SI, reste relativement récente ;

- le fait que les phases de collecte de données doivent nécessairement s'étaler sur des périodes suffisamment longues pour vérifier la pertinence des indicateurs, plus spécifiquement des indicateurs composés - certains SLA ne prennent sens qu'au bout de périodes suffisamment longues ;
- la nécessité de stabiliser les solutions, du simple fait qu'elles dépendent elles-mêmes d'autres API, langages, implémentations ou solutions, qui peuvent eux-même être la source de dysfonctionnements ;
- la mise à niveau des produits *Shinken* et *Canopsis*. Le développement de ces projets étant relativement actif, leur utilisation doit s'accompagner d'une veille effective, plus spécifiquement sur les corrections de *bugs* et les mises à disposition de nouvelles versions. Les montées de version sur les deux produits ont eu un impact notable, particulièrement en ce qui concerne la reprise du paramétrage *Shinken* et les vues *Canopsis* ;
- le fait que le projet ne constituait en aucune façon en la mise à disposition d'une solution « clés en main », mais plutôt en la mise en place d'un processus itératif éprouvé, permettant au service informatique de continuer à étoffer la supervision du SI.

Au bout du compte, il serait peu pertinent de penser l'ensemble comme une solution définitive et immuable. Au contraire - et c'est en ce sens qu'a été engagée la démarche - il s'agissait de fixer les modalités de supervision et d'y intégrer les besoins systématiques d'ajustements et de corrections. L'Illustration 6.1 dans l'Annexe 6.6 schématise les grandes lignes de ce processus. La systématisation du processus de supervision constitue donc un des objectifs de fond du projet. Sa mise en place repose essentiellement sur les phases de test, d'évaluation et de correction.

4.1.2. Difficultés rencontrées

La mise en œuvre du projet s'est heurtée à plusieurs écueils. Certains sont liés à la nature exploratoire de ce travail : essais, expérimentation et codage n'ont pas nécessairement toujours été concluants ou ont nécessité la résolution d'incidents, certains d'entre eux ayant des conséquences graves sur l'infrastructure. S'il existe divers protocoles de supervision, les implémentations proposées par les constructeurs ne sont pas toujours finalisées ou nécessitent certaines précautions. C'est par exemple le cas de l'attribut *ifSpeed* de la MIB-II : Cisco, Nortel ou Netasq lui attribuent des valeurs différentes pour des capacités physiques identiques. Ces difficultés, outre les adaptations qu'elles demandent,

témoignent de l'hétérogénéité des techniques et implémentations en matière de supervision. Si les plates-formes mises en place permettent d'homogénéiser les mesures et le rendu des résultats, d'autres difficultés sont ensuite apparues :

- celles qui sont liées à l'analyse et à l'interprétation du lien entre la vision technique et la vision métier : la restitution d'indicateurs « simplifiés » dans *Canopsis* couvre souvent des architectures logiques complexes. Le fait même de formaliser ces KPI peut mettre à jour des dysfonctionnements jusqu'alors invisibles dans les processus métier ou les architectures techniques. Si les questions techniques peuvent être traitées de façon autonome par le service informatique, l'organisation de GBH présente peu de moyens pour travailler la question du lien entre le métier et la technique. Si un rendu graphique peut être mis à disposition, il ne s'agit pour le moment que d'une *preuve de concept*. C'est également ce qui explique qu'un tel projet ne puisse à ce stade emporter d'autre adhésion que celle des techniciens du SI ;
- celles qui sont liées à la mise en place des composants techniques de la supervision. D'une part, les projets *Shinken* et *Canopsis* demandent nécessairement un certain nombre d'adaptations ou de compléments ; c'est d'ailleurs en ce sens qu'ils ont été conçus. À ce titre, nous examinerons brièvement comment ont été intégrées les alertes passives SNMP et les traces *Syslog*. D'autre part, les sondes techniques proposent souvent une approche générique, dont les mécanismes se révèlent souvent inadaptés aux spécificités du SI de GBH. Aussi leur mise en œuvre nécessite-t-elle un certain nombre de corrections, voire même, dans le cas des sondes EUE, une écriture complète. Enfin, l'infrastructure même de supervision n'est pas sans effet sur la performance du SI. D'un côté, une requête WMI inappropriée, une mauvaise utilisation de certaines sondes ou une implémentation SNMP incomplète[50] peuvent nuire à la performance globale du SI. De l'autre côté, l'infrastructure doit parfois subir des modifications qui nécessitent l'interruption de certains services. C'est notamment le cas lorsqu'il faut recompiler un serveur *NGiNX* pour y intégrer un module *status*[51] ou modifier des règles du *firewall* pour autoriser les flux de supervision en DMZ. Bien que ces interruptions soient minimes, elles doivent être planifiées en fonction de leur impact sur l'activité de production ;
- celles qui sont liées à la veille technologique et aux apprentissages qu'induisent ces solutions de supervision. *Shinken* et *Canopsis* sont des outils qui reposent sur une représentation du SI, elle-même inspirée de modèles. Quoiqu'elles autorisent une

50 Par exemple, les Cisco *Catalyst 3560* renvoient des indicateurs de débit associés au VLAN. Ces indicateurs ne sont pourtant valides que sur certains modèles et versions d'IOS.

51 http://wiki.nginx.org/HttpStubStatusModule

grande liberté, ne serait-ce qu'en permettant l'accès au code source, il n'est clairement pas envisageable d'exploiter ces solutions sans porter une attention particulière à leur évolution et à celle du microcosme qui les entoure - API, protocoles, outils. C'est dans le cadre de cette démarche de veille et d'expérimentation que certaines solutions - comme *Nagvis*, *Thruk* ou *Pnp Nagios* - ont été mises en œuvre, pour être finalement abandonnées faute de pertinence. Cette démarche, bien qu'indispensable et formatrice, demeure extrêmement coûteuse en temps.

Hormis la plate-forme de supervision constituée de deux serveurs, une dernière difficulté tient à la mise en place d'un environnement dédié aux sondes EUE. Dans un premier temps, il avait été envisagé de déporter un *poller Shinken* dédié à ce type de tests sous un système Microsoft Windows. Pourtant, un certain nombre de difficultés sont apparues, principalement liées aux différences de versions entre les modules Python de la bibliothèque *Pyro*. Pour passer outre cette fragilité, une autre maquette a été élaborée sur une machine virtuelle, cette fois-ci en s'appuyant directement sur des mécanismes XML-RPC couplés à des retours passifs vers une instance *WS-arbiter* de *Shinken*. C'est par ce mécanisme qu'il a été possible de piloter les tests EUE via *Sikuli*.

Le projet *Selenium* proposait par ailleurs une méthode de pilotage distant du navigateur *Internet Explorer*. Quoique limité aux navigateurs, *Selenium[46]* a permis de déployer des tests EUE dans un environnement identique à celui des utilisateurs du SI GBH. La véritable difficulté a été d'ordonnancer les tests EUE : *Sikuli* fonctionnant sur la base d'une reconnaissance optique, il est absolument impératif que l'environnement graphique ne soit pas pollué par d'autres interactions, automatisées ou non. De ce fait, deux tests EUE peuvent produire une contention, d'où leur fragilité. La première solution a consisté à mettre en place un verrou logique similaire à un sémaphore, pour éviter que deux tests puissent s'accaparer l'interface graphique simultanément. Cette solution a eu l'inconvénient de créer des situations d'*interblocage*[52] au fur et à mesure que le nombre de tests augmentait. Pour finir, l'utilisation d'un *trigger Shinken* a permis de travailler directement sur l'étalement de l'ordonnancement de ces tests. Beaucoup moins contraignante, l'idée est de vérifier, à chaque lancement d'un test, que le *scheduler Shinken* respecte bien un intervalle de temps minimum entre les différentes exécutions. Si tel n'est pas le cas, on force la valeur du prochain test, comme le montre l'extrait de code du *trigger* :

```
test_queue=get_objects('Seven/check-eue*')
for i in range(len(test_queue)-1):
    if (abs(test_queue[i].next_chk-test_queue[i+1].next_chk))<180:
        test_queue[i+1].next_chk=test_queue[i].next_chk+180
```

52 Un interblocage, encore appelé « étreinte mortelle » ou *deadlocks* est une situation qui se produit en programmation concurrente, lorsque deux processus s'attendent mutuellement de façon infinie.

Au-delà de ces problèmes de mise en œuvre, le véritable écueil consiste à rendre lisible l'état du SI aux utilisateurs concernés. Si l'interface graphique *Canopsis* propose des facilités en ce sens, il reste à élaborer un rendu graphique capable de faire sens pour les utilisateurs. En pratique, les indicateurs techniques - comme les débits, latences ou vitesses - n'ont absolument aucune portée sémantique pour les utilisateurs finaux. L'aboutissement de scénarios de tests comportementaux, puisqu'ils concernent plus directement leur activité, est davantage parlant, mais ne prend en compte que des cas génériques. Les KPI interviennent alors pour pallier à cette difficulté. Leur élaboration nécessite toutefois un travail d'ajustement dont l'objectif est de prendre en compte les enjeux fonctionnels et techniques. À ce stade, si les outils sont mis en place, il reste à affiner ces KPI en collaboration avec les services utilisateurs du SI de GBH. La difficulté sous-jacente à cette démarche, qui se traduit par un très faible intérêt pour les résultats produits, met sans doute en évidence la nécessité d'évaluer de façon plus stricte la criticité du SI par rapport à l'activité de production. La mise en œuvre d'un audit pourrait, par exemple, faire ressortir les points communs entre l'activité de l'organisation et celle du SI. À ce jour, aucune démarche n'a été faite en ce sens, si bien qu'il reste délicat pour les seuls techniciens du SI de dépasser la distinction entre l'outil technique et son usage. Une approche plus collaborative aurait été porteuse de davantage de sens dans l'élaboration des KPI et les possibilités offertes par la plate-forme mise en œuvre autorisent toujours des développements en ce sens. Plus en amont, cela renvoie aussi à la question de la lisibilité des stratégies de l'organisation.

4.2. Techniques de collecte de données

Nous observerons ici quelques exemples des implémentations qui ont permis de réaliser la collecte de données. On distingue donc deux types de sondes :

- les sondes techniques, qui testent les différents composants de la plate-forme technique ;
- les sondes comportementales, qui émulent le comportement d'un utilisateur.

4.2.1. Sondes techniques

Le principe des sondes techniques a été décrit plus haut au paragraphe 3.1.1.2. En pratique, une vingtaine de *packs* regroupant des ensembles de sondes techniques ont été mis au point. Un grand nombre de sondes sont mises à disposition par la communauté[78]. Toutefois, beaucoup d'entre elles ne sont pas toujours adaptées aux spécificités du SI de GBH. C'est à ce niveau qu'un grand nombre de corrections ont été nécessaires.

4.2.1.1. Techniques mises en œuvre

En se reposant sur le module Python *PySNMP[114]* ainsi que sur des scripts en langage Shell ou Perl, des sondes ont été intégralement réécrites ou corrigées.

Il s'agit principalement :

- de sondes pour les routeurs Cisco et le *firewall* Netasq U250 ;
- de scripts destinés à l'exécution distante via NRPE, notamment pour les serveurs *Citrix* dont le fournisseur SNMP est sous une licence qui ne nous était pas accessible.

Pour le reste, un grand nombre de modifications ont été ajoutées à des *plug-ins* existants, essentiellement dans l'objectif d'homogénéiser les mesures de performance, ou de créer des mesures de performance complémentaires. Il a par exemple été difficile d'obtenir des unités de mesures comparables. Dans le cas des sondes réseau, les unités de mesures sont rarement homogènes. On trouve selon les implémentations des usages très divers : *bit/seconde*, *bytes/seconde*, pourcentage de bande passante, *frames* par seconde, etc. Dans notre cas, l'intégralité des mesures a été corrigée de façon à renvoyer systématiquement des mesures en octets (*bytes*) de façon à faciliter la comparaison des débits en volume.

Au-delà du SNMP, les principaux protocoles mis en œuvre sont :

- WMI pour les systèmes Microsoft Windows à l'aide du programme Perl *check_wmi_plus[42]*;
- l'invocation directe de clients via des scripts Shell par exemple :
    - *Smicli* pour obtenir des mesures de capacité du SAN IBM DS3400,
    - l'API *VMware* via le programme Perl *check_vmware _api*[53],
    - l'API *XenServer* via un script Perl développé par la société OP5[54],
    - une émulation Java de *terminal 3270* pour récupérer les informations auprès de l'iSeries OS/400,
    - des scripts Shell pour inventorier les événements des logs des serveurs SMTP.

Les techniques d'obtention d'informations sont donc très variées. Elles ne reposent pas nécessairement sur des protocoles, mais davantage sur des techniques d'automatisation : automatisation de requêtes, simulation d'accès à des interfaces d'administration, sollicitation d'API spécifiques, etc. En tout état de cause, le seul recours au protocole SNMP ou WMI n'est pas suffisant.

53 OP5 Plugins : https://kb.op5.com/display/PLUGINS/Plugins+Home

54 Idem sur le dépôt git OP5 :http://git.op5.org/git/?p=nagios/op5plugins/check_xenapi.git

4.2.1.2. Exemples de sondes

Avec la bibliothèque *PySNMP*, les scripts peuvent suivre le même synopsis. Nous nous intéressons ici à une sonde dont le but est d'évaluer le débit d'interface réseau donné en paramètres. Les mesures reposent sur la MIB *IF-MIB*. Cette MIB fournit une traduction littérale des OID et facilite le travail sur la sémantique des variables. Le chargement de la MIB n'est donc pas obligatoire.

On récupère les arguments principaux : adresses IP, communauté SNMP et dénomination du périphérique réseau :

```
 def arg_parse():
     usage="%prog [options] arg"
     parser = OptionParser(usage)
     parser.add_option("-H", "--hostname", dest="hostname",help="device to query", default=
'localhost' )
     parser.add_option("-C", "--community", dest="community",help="snmp community",default='ghbro')
     parser.add_option("-D", "--device", dest="device",help="device name ifName netasq  Ethernet0
Ethernet1 etc.. ",default='eth0')
     (options, args) = parser.parse_args(args=sys.argv[1:])
     if len(args)!=0:
         parser.error("Error in command line options")

     return options
```

On émet ensuite une requête SNMP sur l'OID cible. On parcourt les résultats de la table, qu'on renvoie au retour de la fonction *getOidval* :

```
 def getOidval(myoid,community,ip):
     # Create MIB loader/builder
     mibBuilder = builder.MibBuilder()
     mibSources = mibBuilder.getMibSources() + (
         builder.DirMibSource('/usr/local/shinken/var/MIBS/python'),
         )
     mibBuilder.setMibSources(*mibSources)
     mibBuilder.loadModules(
          'IF-MIB','NAS'
         )
     mibViewController = view.MibViewController(mibBuilder)
     cmdGen = cmdgen.CommandGenerator()
     errorIndication, errorStatus, errorIndex, varBindTable = cmdGen.bulkCmd(
         cmdgen.CommunityData(str(community)),
         cmdgen.UdpTransportTarget((str(ip), 161)),
         0,25,
         myoid,
         lookupValues=True,
         )
     if errorIndication:
             print(errorIndication)
     else:
         if errorStatus:
             print('%s at %s' % (
                     errorStatus.prettyPrint(),
                     errorIndex and varBindTable[-1][int(errorIndex)-1] or '?'
                     )
                 )
         else:
             mydata={}
             for varBindTableRow in varBindTable:
                 for name, val in varBindTableRow:
                     (oid, label, suffix  )= mibViewController.getNodeName(name)
                     idx=len(label)-1
                     objname=label[idx]
                     objname = str(objname)+'.' + str(suffix)
                     mydata[objname]=val.prettyPrint()
             return mydata
```

Le script principal émet deux requêtes pour récupérer, par exemple, les valeurs de l'objet *ifHCOutOctets*. Celui-ci nous indique le niveau de sortie de l'interface réseau en octets. Les valeurs du dernier *check* sont inscrites dans un fichier à l'aide du module *pickle*[55]. De cette façon, on peut comparer les deux volumes sur une période de temps précise :

```
 deltatime=int(obj['ts'])-int(olddata['ts'])
     delta_inoctet = abs(int(obj['ifHCInOctets'])-int(olddata['ifHCInOctets']))
     delta_outoctet = abs(int(obj['ifHCOutOctets'])-int(olddata['ifHCOutOctets']))
     if deltatime>0:
         debitIn=delta_inoctet/deltatime
         debitOut=delta_outoctet/deltatime
     #pourcentages
     bandepass= FDBandwith(delta_inoctet,delta_outoctet,deltatime,obj['ifHighSpeed'])
     outUse=outputUse(delta_outoctet,deltatime,obj['ifHighSpeed'])
     inUse=inputUse(delta_inoctet,deltatime,obj['ifHighSpeed'])
     #warn= 80% => 0.8 * 104857600 = 83886080
     #crit=95% 0.95 * 10857600 = 99614720
     #print(" ecart de %s depuis le dernier check",str(deltatime))

     writeData(filename,obj)
```

Le calcul de la bande passante doit prendre en compte la vitesse maximale indiquée par la valeur de l'item *ifSpeed*. Toutefois, selon les matériels, cette vitesse ne suit pas les mêmes nomenclatures.

Par exemple, les instances *XenServer* renvoient un objet *ifSpeed* à 0 si les cartes sont au *Gigaoctet/sec*. Ci-dessous, la fonction de calcul de la bande passante prend en compte cette exception :

```
 def FDBandwith(DeltaIfino,DeltaIfouto,Deltasec,ifSpeed):
    #only xen VMs
    if ifSpeed==0:ifSpeed=1000
    if Deltasec==0:return 0
    if ifSpeed==10000:speed=ifSpeed * 1000000 /8 #carte 10Gb/sec
    if ifSpeed==1000:speed=ifSpeed * 1000000 /8
    if ifSpeed==100:speed=ifSpeed * 1000000 /8
    if ifSpeed==10:speed=ifSpeed * 1000000 /8
    Octetdeb=max(int(DeltaIfino),int(DeltaIfouto)) / Deltasec
       deb  =  float  (((max(float(DeltaIfino),float(DeltaIfouto))  *  100  )  /  (Deltasec)  )    /
(  speed ) )
    debformated= '%.4f'%deb
 #   print("test : %s" % test)
    return debformated
```

Autre exemple : sur un matériel de type Cisco, les compteurs demandent une évaluation différente de l'objet *ifSpeed* :

```
 def FDBandwith(DeltaIfino,DeltaIfouto,Deltasec,ifSpeed):
    if Deltasec==0:return 0
    if ifSpeed==1000:speed=ifSpeed * 1000000 /8
    if ifSpeed==100:speed=ifSpeed * 1000000 /8
    if ifSpeed==10:speed=ifSpeed * 1000000 /8
```

55 https://docs.python.org/2/library/pickle.html

Enfin, une fois les calculs d'indicateurs de performance effectués, on réécrit la valeur dans un fichier tampon, via le module Python *pickle*, puis on renvoie le résultat comprenant l'intégralité des mesures de performances :

```
      print("OK - "+my_device+"  ("+str(obj['ifAlias'])+" Debit:"+str(bandepass)+"% In:"+str(inUse)
+"% Out:"+str(outUse)+"% |BandePassante="+str(bandepass)+"%;0;0;75;100
InUse_in_KBs="+str(debitIn/1024)+"KB;0;0;0;0 OutUse_in_KBs="+str(debitOut/1024)+"KB;0;0;0;0
ifHCInUcastPkts="+str(obj['ifHCInUcastPkts'])+";0;0;0;0
ifHCOutUcastPkts="+str(obj['ifHCOutUcastPkts'])+";0;0;0;0
ifHCOutBroadcastPkts="+str(obj['ifHCOutBroadcastPkts'])+";0;0;0;0
ifHCInBroadcastPkts="+str(obj['ifHCInBroadcastPkts'])+";0;0;0;0")
      sys.exit(0)
```

En langage Perl, la logique est sensiblement la même. Au lieu de solliciter les méthodes de la bibliothèque *PySNMP*, on s'appuie sur le module Perl Net::SNMP. Partant d'OID connus, en réalité des index de tables, on peut ainsi récupérer l'ensemble des valeurs des sous-branches. On crée une session SNMP en invoquant une fonction qui renvoie le contexte de cette session. Par exemple :

```
 sub _create_session {
       my ($server, $comm) = @_;
       my $version = 1;
       my  ($sess,  $err)  =  Net::SNMP->session(  -hostname  =>  $server,  -version  =>  $version,
-community => $comm);
       if (!defined($sess)) {
              print "Can't create SNMP session to $server\n";
              exit(1);
       }
       return $sess;
 }
```

À partir de là, on récupère les informations souhaitées, ici la température interne d'un *switch* :

```
       my $temp;
         my $S_temp = ".1.3.6.1.4.1.9.9.13.1.3.1.3";
       my $R_tbl = $snmp_session->get_table($S_temp);
       foreach my $oid ( keys %$R_tbl) {
              $temp = "$$R_tbl{$oid}";
              last;
       }

       if("$temp" eq "") {
              print "The switch $switch can't report temperature via SNMP\n";
              FSyntaxError();
       }

       if($temp > 1) {
              if($warn > $crit and "$check_type") {
                     print "Warning can't be larger then Critical: $warn > $crit\n";
                     FSyntaxError();
              }
              if($temp <= $warn) {
                     $stat = 0;
                     $msg = "Temperature: OK - Tempeture is $temp Celsius";
              }  elsif($temp > $warn and $temp < $crit) {
                     $stat = 1;
                     $msg = "Temperature: Warn - Tempeture is $temp Celsius";
              } elsif($temp >= $crit) {
                     $stat = 2;
                     $msg = "Temperature: CRIT - Tempeture is $temp Celsius";
              }
              $perf = "temperature=$temp;$warn;$crit";
```

Lorsque l'on ne dispose d'aucun protocole de supervision, on peut mettre en œuvre des techniques d'analyses syntaxiques : expressions régulières, analyse textuelle en fonction de la position dans un écran de terminal, etc.

Par exemple, pour vérifier le statut d'un routeur ADSL d'appoint, outre les règles de routage du *firewall*, on peut accéder relativement facilement à une page indiquant le statut des connexions. En Python, on peut extraire les valeurs pertinentes avec une fonction comme celle décrite ci-dessous : on récupère la page de statut par HTTP, puis on analyse son contenu, qu'on structure dans un dictionnaire.

```
def parse_status_page():
    """Parser freebox Statut"""
    file = open('/usr/local/shinken/var/freeboxx-status.log', 'r')
    line = file.readline()
    dictStatus = {}
    counter = 1

    while line:
        if "bit ATM" in line:
            dictStatus['debitATMDescinkbps']=line.split()[2]
            dictStatus['debitATMAscinkbps']=line.split()[4]
        elif "WAN" in line:
            dictStatus['debitWANInkopersec']=line.split()[2]
            dictStatus['debitWANOutkopersec']=line.split()[4]
        elif "Switch" in line:
            dictStatus['debitSwitchInkopersec']=line.split()[2]
            dictStatus['debitSwitchOutkopersec']=line.split()[4]
        line = file.readline()

    return dictStatus
```

Globalement, seules les modalités d'obtention des informations changent. Il faut en outre rester attentif à la performance de ces sondes : une sonde trop complexe risque d'augmenter la latence de l'ordonnanceur *Shinken*. Certaines techniques d'analyse de fichiers journaux induisent, par exemple, trop de latence entre l'envoi du test et l'obtention du résultat. En règle générale, il convient de réduire au maximum le délai entre le moment où la tâche est lancée par l'ordonnanceur et le moment où ce dernier traite le résultat. Dans le cas contraire, les performances du serveur se détériorent avec l'augmentation de la charge système.

4.2.1.3. Rendu graphique avec Graphite

Outre la transmission des événements de supervision vers *Canopsis*, *Shinken* permet, toujours par l'intermédiaire d'un module, d'exporter les données de performance vers un gestionnaire de graphique dénommé *Graphite*. Les données de performance associées aux estampilles temporelles sont reconstituées comme des coordonnées.

Graphite est constitué d'un daemon *carbon-cache*[56] chargé d'intégrer les données que lui fait parvenir le *broker Shinken* dans une base de données *whisper* spécialisée dans le stockage des séries temporelles.

56 http://graphite.readthedocs.org/en/latest/carbon-daemons.html

Illustration 4.2 : le backend « Graphite »

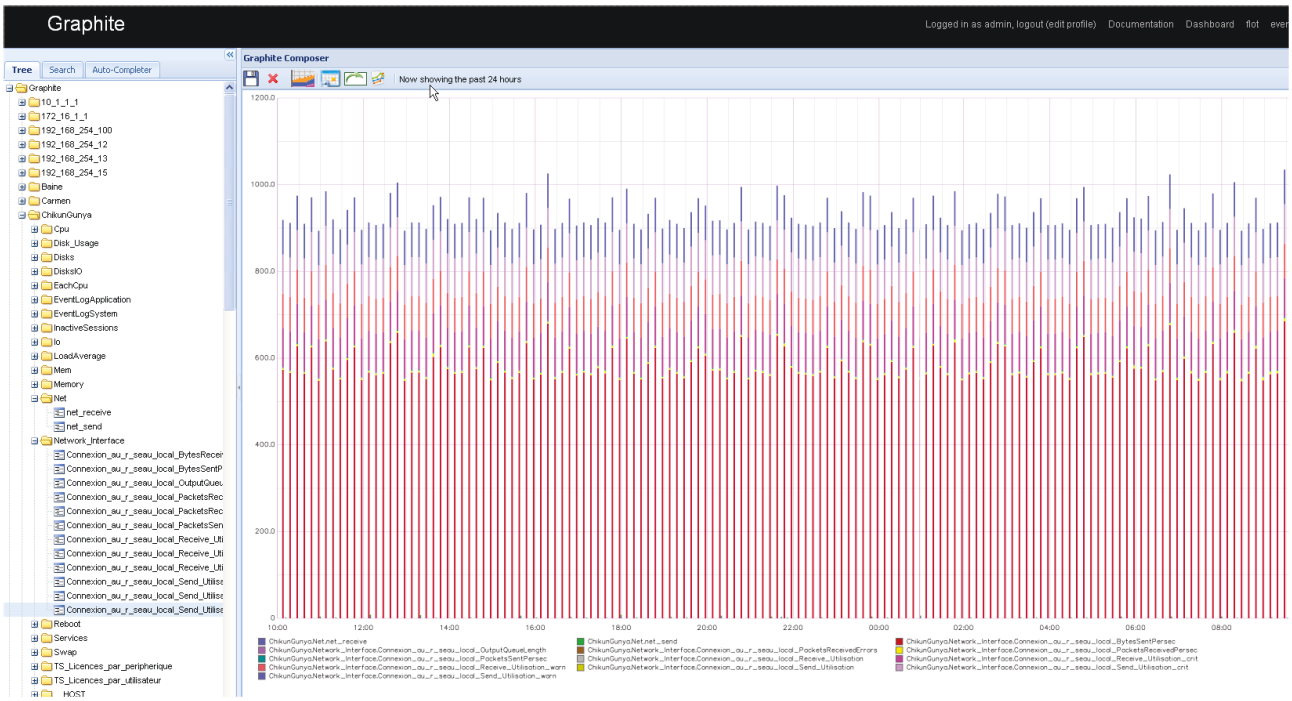


Illustration 4.3 : l'interface web de Graphite

L'uniformisation des données de performance est assurée par le respect du formalisme de sortie préconisé par *Nagios* et par le module *broker Graphite*. Une fois collectées, les données peuvent être ré-exploitées et manipulées via une interface graphique web. L'application web, basée sur le *framework* Python *Django*[57], permet d'effectuer cer-

57 http://www.django-fr.org/

taines opérations sur ces données temporelles : agrégation, composition de tableaux de bord. Les graphiques des données de performance sont récupérés par *Shinken*, qui les intègre dans sa propre interface graphique. Les Illustration 4.3 et Illustration 4.4 précisent ces deux moyens d'exploiter les données de performance. De nombreux autres produits exploitent les *rrdtools*[58]. Ces derniers imposent néanmoins un certain nombre de contraintes que l'auteur de *Whisper* a souhaité lever dans son implémentation :

> *RRD is great, and initially Graphite did use RRD for storage. Over time though, we ran into several issues inherent to RRD's design.*
>
> - *RRD can't take updates for a timestamp prior to its most recent update. So for example, if you miss an update for some reason, you have no simple way of back-filling your RRD file by telling rrdtool to apply an update to the past. Whisper does not have this limitation, and this makes importing historical data into Graphite way way easier.*
> - *At the time Whisper was written, RRD did not support compacting multiple updates into a single operation. This feature is critical to Graphite's scalability.*
> - *RRD doesn't like irregular updates. If you update an RRD but don't follow up another update soon, your original update will be lost. This is the straw that broke the camel's back, since Graphite is used for various operational metrics, some of which do not occur regularly (randomly occuring errors for instance). We started to notice that Graphite sometimes wouldn't display data points which we knew existed because we'd received alarms on them from other tools. The problem turned out to be that RRD was dropping the data points because they were irregular. Whisper had to be written to ensure that all data was reliably stored and accessible.[115]*

58 http://oss.oetiker.ch/rrdtool/

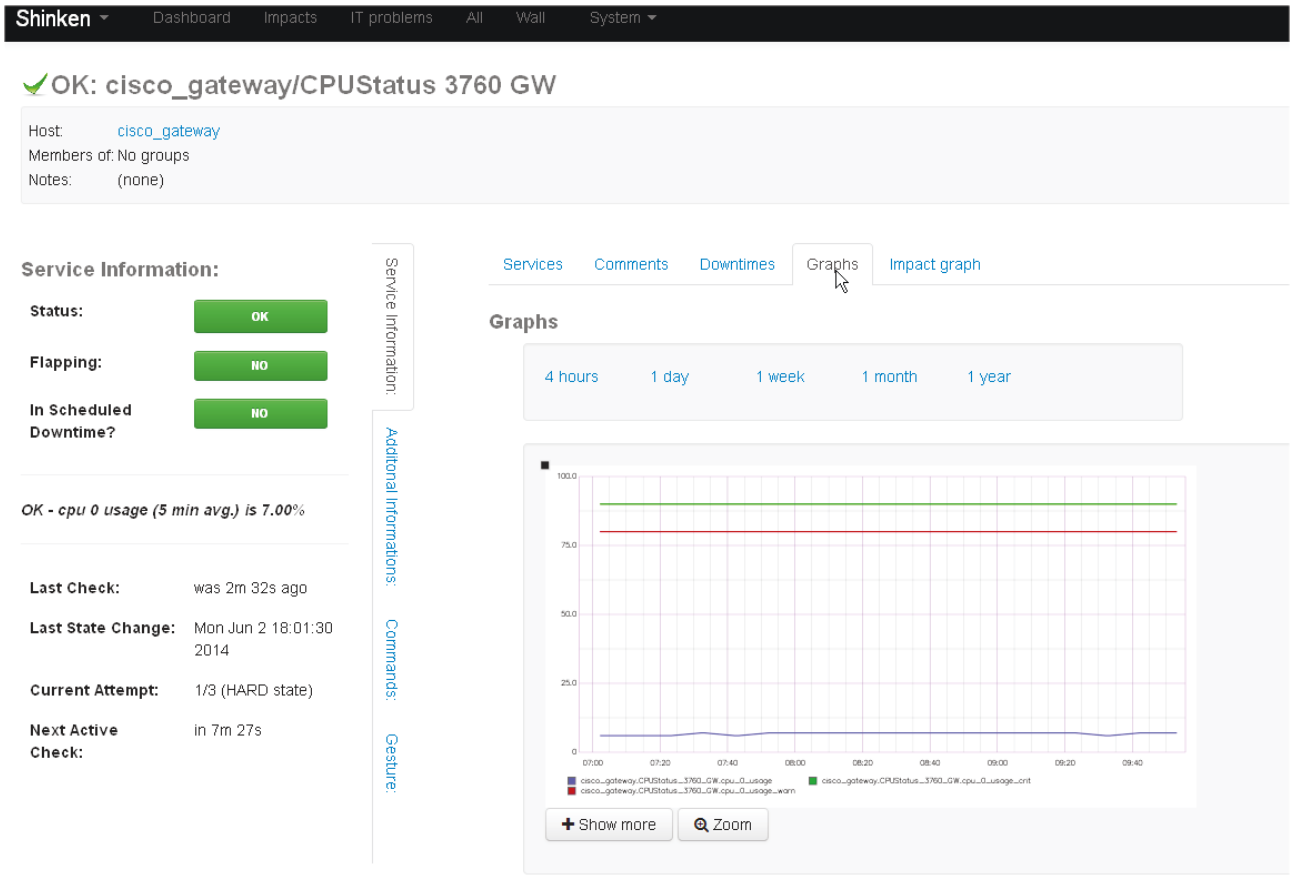


Illustration 4.4 : exploitation du moteur Graphite pour les graphiques Shinken

En matière de rendu, *Graphite* propose une API de rendu graphique que l'on sollicite par l'intermédiaire d'une URL HTTP incluant les paramètres adéquats[116]. Cette API permet de systématiser le rendu des graphiques directement dans l'interface web *Shinken*, ainsi que le montre l'Illustration 4.4.

4.2.2. End User Experience : les tests comportementaux

Les tests techniques intéressent davantage les techniciens tels que les administrateurs système et réseau, plutôt que les responsables fonctionnels. Pourtant, ces derniers ont régulièrement besoin de savoir si l'application sur laquelle repose leur activité fonctionne « normalement », par exemple avec des temps de réponse acceptables. Le point de vue du *ressenti utilisateur* n'est pas exclusif : il doit être couplé aux résultats fournis par les sondes techniques. À contrario, l'élaboration de scénarios comportementaux s'exécutant conformément à un synopsis établi garantit la disponibilité et la possibilité d'évaluer la performance de l'application. Les procédés de tests comportementaux évoqués dans le paragraphe 2.3.1.4 sont mis en œuvre notamment pour fournir à des responsables fonctionnels qui ne disposent d'aucune connaissance technique un moyen d'évaluer une application.

Dans le cadre du projet, diverses techniques ont été mises en œuvre. Nous nous proposons d'en exposer deux. La première consiste à simuler un scénario à l'aide d'un navigateur *headless*, comme le permettent les solutions *Selenium* ou *PhantomJS*. La seconde a consisté à utiliser l'API Jython de la solution *Sikuli* pour évaluer des scénarios imposant l'usage d'interfaces graphiques « lourdes ».

4.2.2.1. Tests EUE avec CasperJS : l'exemple de l'interface imhoweb

Imhoweb est une solution de gestion de la demande de logement réalisée par la société Sigma. La gestion de la demande de logement a récemment été externalisée vers une plate-forme gérée par cette société. Un des principaux problèmes engendrés par cette externalisation est justement de donner aux responsables fonctionnels le moyen de vérifier que l'interface est accessible via le réseau internet et qu'elle est en mesure de répondre à un scénario fonctionnel simple.

Dans ce contexte, outre les indicateurs techniques courants (ping, vérification de l'accès HTTP, etc.), une sonde fonctionnelle a été élaborée en vue de simuler le comportement d'un utilisateur à l'aide de l'API *CasperJS[117]*.

La sonde est écrite en JavaScript. Le principe consiste à exécuter un scénario - soit une suite d'étapes - à l'aide d'un navigateur sans interface, que l'on pilote de façon programmatique. Le navigateur *aveugle (headless)* repose sur l'API *Webkit QT*. Le code du programme s'organise en plusieurs étapes.

- Une instanciation du navigateur aveugle :

```
 #!/usr/bin/casperjs
 var casper = require('casper').create({
       pageSettings: {
         loadImages:  true,
         loadPlugins: true,
    },
    verbose: true,
    logLevel: 'debug'
 }
 );
 //special func
 var links = document.getElementsByTagName('a');
 links = Array.prototype.map.call(links,function(link){
        return link.getAttribute('href');
 });
 var last_timestamp, started_at;
 last_timestamp = started_at  = Date.now();
 var perf_data = [];
 var failures = [];
 var x = require('casper').selectXPath;
 casper.on('starting', function() {
               this.echo("Browser ready and starting at " + Date.now());
               elapsed("Browser ready");
 });

 casper.start('http://www.fdls-bourgognefranchecomte.fr/imhoweb21/');
 casper.userAgent('Mozilla/5.0 (Windows; U; Windows NT 6.1; fr; rv:1.9.0.6) Gecko/2009011913
Firefox/3.0.6');
```

- L'exécution d'une étape. Ici, l'étape de connexion à l'interface imhoweb :

```
casper.then(function() {
                timerize(elapsed("login", true));
                this.page.switchToChildFrame(0)
                this.fill('form[name=FmFormulaire]',{
                     'EdNomConnexion': "BAUDIQUEY",
                     'EdPrenomConnexion' : "Renaud",
                            'EdPWD' : "xxxxx"
                           }, true);
                this.click("#aze");
});
```

- Chaque étape du scénario est assortie d'une évaluation de sa durée d'exécution. L'aboutissement d'une étape est conditionné par la recherche d'éléments du DOM de la page HTML :

```
casper.then(function(){
                  timerize(elapsed("recherche_dde", true));
                  this.page.switchToChildFrame('framePrincipale');
                  this.waitFor(function check() {
                     return this.evaluate(function() {
                         return document.querySelectorAll('#champsRecherche');
                     });
                }, function then() {
                     this.page.switchToChildFrame('framePrincipale');
                     this.capture('out.png');
                     this.fill('form[name=FmFormulaire]',{
                     'EdNom' : "DEN"
                  },false);
                });

});
```

- La sonde exécute donc l'ensemble des étapes de façon ordonnée puis fournit un résultat au format *Nagios* :

```
casper.run(function() {

        perf_data.unshift(elapsed("Total"));

               // on completion of all the steps, exit "OK"
               var result = "OK";
               var message = "IMHOWEB TESTING";

               nagios_exit(
                   "CHECK_SITE_IMHOWEB",
                   result,
                   message,
                   perf_data.join(' ')
               );
           });

function elapsed(label, reset) {
    var now = Date.now();
    var elapsed_ms;

    if (reset) {
        elapsed_ms = now - last_timestamp;
        last_timestamp = now;
            }
    else {
        elapsed_ms = now - started_at;
    }
    // casper.echo(label + ': ' + elapsed_ms.toString() + " ms elapsed." );
    return "'" + label + "'=" + elapsed_ms + 'ms';
}

function timerize(p) {
    perf_data.push(p);
```

```
}


function fail(desc) {
    casper.capture('fail.png');

    nagios_exit(
        "CHECK_SITE_IMHOWEB",
        'CRITICAL',
        "IMHOWEB SCNEARIO fails: " + desc + ". Screenshot saved to 'fail.png'.",
        perf_data.join(' ')
    );
}
```

Le résultat est pris en compte dans *Shinken*, puis transmis à *Canopsis,* où les données de performance peuvent faire l'objet d'un rendu graphique :

Illustration 4.5 : rendu graphique d'un scénario EUE

4.2.2.2. Tests EUE avec Sikuli

L'interface *Sikuli*, comme il a été évoqué plus haut, peut être pilotée par le langage Python. *Sikuli* embarque une implémentation de Python en Java, Jython (cf. Annexe 6.7). Son usage est toutefois lié à quelques restrictions, notamment en ce qui concerne l'inclusion de modules Python externes. Nous avons choisi d'utiliser une implémentation du protocole XML-RPC[118] pour permettre au serveur *Shinken* de lancer des tests à distance sur une station de travail utilisateur. Le serveur XML-RPC (la station de travail) met à la disposition du client une série de procédures, qui permettent de lancer le programme de simulation étape par étape depuis le serveur de supervision *Shinken*. Lors de l'exécution d'une procédure-scénario, le programme serveur va émettre des *checks* passifs vers le module *WS-arbiter* de *Shinken*, de façon à récupérer les résultats de chaque étape du déroulement du scénario. Le principe consiste à découper le scénario fonctionnel en étapes distinctes et à mesurer leur durée d'exécution. En principe, le non-aboutissement d'une de ces étapes entraîne une invalidation du test. Dans le cas contraire, on récupère la durée d'exécution de chaque étape.

Dans le code, on définit une fonction *pushmessage* capable d'envoyer un message passif à *Shinken* tout en continuant le déroulement du scénario :

```
def pushmsg(mymsg,svcname):
# format du message OK : step exit |step_exit_duree_lancement=8888.2
    resp=mymsg.split(':')[0].strip()
    print resp
    msg=mymsg.split('|')[0].split(':')[1].strip()
    print msg
```

```
     perfdata=mymsg.split('|')[1].strip()
     print perfdata
     if resp=='OK':rc=0
     else:rc=2
     timestamp=time.time()
     hostname="Seven"
     servicedesc=svcname
     #send data to web service
     curl = pycurl.Curl()
     curl.setopt(curl.URL, "http://127.0.0.1:7760/push_check_result")
                curl.setopt(curl.POSTFIELDS,"timestamp="+str(timestamp)+"&host_name="+str(hostname)
+"&service_description="+str(servicedesc)+"&return_code="+str(rc)+"&output="+str(msg)
+"|"+str(perfdata))
     curl.perform()
     curl.close()
     return rc
```

On exécute les appels XML-RPC pour chaque étape du scénario : ici, on simule l'utilisation de l'interface lourde de l'ERP *Aravis* pour rechercher une facture dans le module comptabilité. Le test se décompose en 4 étapes *(steps)* : lancement de l'interface, identification et connexion, recherche d'une facture et vérification du résultat, sortie de l'interface.

```
# main : lancement client xml-rpc
s = ServerProxy("http://172.16.1.90:1337/")
try:
     mymsg=s.launcharavis()
     initrc=0
     inittime=time.time()
     initmsg="suite de test aboutie"
     rc=pushmsg(mymsg,'sikuli-check-aravis-launch')
     mymsg=s.connect_aravis()
     if rc>initrc:
         initrc=rc
         initmsg+=" step lancement :  "+str(mymsg.split('|')[0].split(':')[1].strip())
     rc=pushmsg(mymsg,'sikuli-check-aravis-connect')
     if rc>initrc:
         initrc=rc
         initmsg+=" step connexion :  "+str(mymsg.split('|')[0].split(':')[1].strip())
     mymsg=s.rechercher_facture()
     rc=pushmsg(mymsg,'sikuli-check-aravis-recherchefacture')
     if rc>initrc:
         initrc=rc
         initmsg+=" step recherche fac :  "+str(mymsg.split('|')[0].split(':')[1].strip())
     mymsg=s.exit_aravis()
     rc=pushmsg(mymsg,'sikuli-check-aravis-exit')
     if rc>initrc:
         initrc=rc
         initmsg+=" step exit :  "+str(mymsg.split('|')[0].split(':')[1].strip())
```

Côté serveur, on utilise un code Python qui pourra être exécuté par l'interpréteur Jython de *Sikuli* et dont suivent des extraits :

- On définit une instance serveur XML-RPC :

```
Settings.UserLogs=False
#Import des classes externes
import sys
myLib="c:\\sikuli\\prog\\xmlrpc.sikuli\\"
if not myLib in sys.path:
    sys.path.append(myLib)

sys.path.append("c:\\sikuli")
from sikuli.Sikuli import *
```

```
import time
import pickle
import inspect

from SimpleXMLRPCServer import SimpleXMLRPCServer as Server
from SimpleXMLRPCServer import SimpleXMLRPCRequestHandler

class SikuliServer(Server):
    #server loop
    def serve_forever(self):
        self.quit=0
        while not self.quit:
            self.handle_request()
        print "request to exit"
        sys.exit(0)
[...]
```

- Les étapes du test sont définies dans des fonctions spécifiques. À ce niveau, on sollicite l'API *Sikuli* (fonctions : *wait app.Open*, *click* etc.). Les instructions de type *click(« xxxx.png »)* sollicitent une comparaison par reconnaissance optique entre une image capturée préalablement et l'affichage de l'interface :

```
def step_launcharavis():
    debut_test=time.time()
    debut_lancement=time.time()
    global sAravis
    sAravis=App.open("u:\\Aravis\\Aravis.exe")
    try:
        wait("1390382069745.png",10)
    except:
        fin_lancement=time.time()
        difflancement="%.1f"%(fin_lancement-debut_lancement)
        return 'KO : probleme lancement aravis not found |'+ "step_launch_duree_lancement=0"
    fin_lancement=time.time()
    difflancement="%.1f"%(fin_lancement-debut_lancement)
    return 'OK : step 1 lancement |'+ "step_launch_duree_lancement="+str(difflancement)

def step_connectaravis():
    debut_test=time.time()
    debut_lancement=time.time()
    try :
        click("1390383332731.png")
        type("CORRESPONDANT")
        click("1390383361003.png")
        type("xxxx")
        click("1390383383363.png")
    except FindFailed:
        sAravis.close()
        #App.open("C:\\selenium\\cleansession.bat")
        fin_lancement=time.time()
        difflancement="%.1f"%(fin_lancement-debut_lancement)
        return 'KO : login aravis can not login | '+" step_connect_duree_lancement=0"
    fin_lancement=time.time()
    difflancement="%.1f"%(fin_lancement-debut_lancement)
    return 'OK : step connect lancement |'+ "step_connect_duree_lancement="+str(difflancement)
[...]
```

- Une fois cela fait, on instancie le serveur XML-RPC qui va exposer les différentes étapes comme des procédures distantes :

```
try:
    srv = SikuliServer(("172.16.16.68",1337))
    srv.register_function(step_skeleton)
    srv.register_function(ext,"quit")
    srv.register_function(step_launcharavis,"launcharavis")
    srv.register_function(step_connectaravis,"connect_aravis")
    srv.register_function(step_exit_aravis,"exit_aravis")
    srv.register_function(step_rechercher_facture,"rechercher_facture")
```

```
    srv.serve_forever()
except:
    print "Erreur Serveur : exit"
    sys.exit(1)
```

Ce procédé d'appel de procédures distantes est indispensable lorsque l'on ne peut utiliser le serveur de supervision comme serveur de test. Les systèmes Linux sont en mesure de mettre en œuvre une interface *Xvfb*[59], mais il reste plus délicat d'interagir par ce biais sur un système Windows. Les tests *Sikuli* sont concluants mais restent fragiles : dès lors qu'une autre application interagit avec le bureau Windows, les procédés de reconnaissance optique sont parasités.

Plus généralement, il est assez simple d'insérer un envoi de message passif vers *Shinken* (via le module *WS-arbiter*), ou directement vers *Canopsis* : les API web peuvent être sollicitées par l'utilisation de l'API cURL[60], qui permet de forger une requête HTTP. Dès lors que cela est possible, c'est-à-dire pour un code techniquement et légalement modifiable, on peut insérer des requêtes pour superviser certaines actions des utilisateurs. Ce procédé, mis en évidence dans le modèle APM[61], intervient en complément des tests comportementaux. Son usage reste toutefois limité en raison des contraintes d'implémentation évoquées plus haut.

4.2.3. Prise en compte des messages SNMP passifs

Bien que *Shinken* puisse prendre en compte des alertes SNMP passives, cela comporte une difficulté inhérente au mode SNMP passif. En effet, ainsi que nous l'avons vu plus haut, chaque constructeur peut implémenter sa propre nomenclature d'alertes au moyen d'une MIB. Sans cette MIB, le message d'alerte passif est difficilement lisible ou interprétable. *Shinken* permet de prendre en compte les alertes passives sur un hôte, à l'aide de la clause de configuration *« passive_checks_enabled »*, mais la collecte de ces alertes et leur classification peut poser problème. D'une part toutes les *traps* SNMP ne sont pas nécessairement pertinentes et d'autre part, leur exploitation repose sur la possibilité de les interpréter de façon littérale. De fait, l'activation des *« passives checks »* dans *Shinken* permet effectivement d'intercepter ces *« traps »* et éventuellement de les notifier, mais pas d'en établir une vision synthétique et organisée. Xavier Dusart présente les modalités de mise en œuvre de ce type d'interception dans un article consacré à la capture des *traps* SNMP[119]. De cette façon, en s'appuyant sur le programme SNMPTT[120], il est envisageable de rediriger les alertes vers un programme qui génère

59 Xvfb − virtual framebuffer X server : http://www.x.org/archive/X11R7.7/doc/man/man1/Xvfb.1.xhtml

60 cURL est un *framework* particulièrement utilisé pour automatiser les accès à des urls distantes. Voir : http://curl.haxx.se/

61 Voir l' Illustration 2.16

un événement vers *Shinken*. L'alerte SNMP est transformée en *check*, pour peu que la chaîne du message SNMP soit correctement interprétée par le *daemon SNMPTT*. La véritable difficulté est de pouvoir analyser, éventuellement avec des expressions régulières, chaque type d'alerte. Le procédé, très générique, ne permet finalement pas d'établir de façon satisfaisante un suivi historisé et classifié de ces alertes. Pour cette raison, les alertes passives n'ont pas été mises en œuvre dans *Shinken*, mais directement redirigées vers l'instance *Canopsis*. L'objectif étant de pouvoir effectuer un suivi de ces alertes et de pointer une corrélation entre les alertes passives et le comportement de certains hôtes ou services, il a semblé plus pertinent de transformer directement les alertes passives en messages *Canopsis* de type *trap*.

Les instructions d'interception SNMP dans la configuration du *daemon* SNMPTT sont modifiées comme sur cet extrait de configuration :

```
  EVENT linkDown .1.3.6.1.6.3.1.1.5.3 "Status Events" Normal
  FORMAT Link down on interface $1.  Admin state: $2.  Operational state: $3
  #EXEC qpage -f TRAP notifygroup1 "Link down on interface $1.  Admin state: $2.  Operational state:
$3"
  EXEC /var/lib/shinken/libexec/push-passive-checks.py "$aA" "$A"  "$c" "$D" "$N"  "$s"
  SDESC
  A linkDown trap signifies that the SNMP entity, acting in
  an agent role, has detected that the ifOperStatus object for
  one of its communication links is about to enter the down
  state from some other state (but not from the notPresent
  state).  This other state is indicated by the included value
  of ifOperStatus.
  EDESC
```

Le programme *« push-passive-checks »* de la clause EXEC assemble les informations en un message *Canopsis* :

```
  […]
  msg="<b>host   :   </b>"+str(hostname)+"</b><br   /><b>severity   :   </b>"+str(severity)+"   <br
/><b>description :</b> "+str(descr)+"<br/> "
  timestamp=time.time()
  idressource='getTrap'+str(oid)
  # Configurations
  host = "172.16.1.47"
  port = 5672
  user = "guest"
  password = "guest"
  vhost = "canopsis"
  exchange = "canopsis.events"

  event = {
      "timestamp":        int(time.time()),
      "connector":        "snmp",
      "connector_name":   "PassiveChecks",
      "event_type":       "trap",
      "source_type":      "trap SNMP",
      "component":        hostname,
      "resource":     idressource,
      "state":        0,
      "state_type":       1,
      "output":       msg,
      "display_name":     oid,
      "snmp_severity": severity,
      "snmp_state": severity,
```

```
    "snmp_oid":oid,
    "address":  hostname
 }

 routing_key    =    "%s.%s.%s.%s.%s.%s.%s"    %    (event['connector'],    event['connector_name'],
event['event_type'],                    event['source_type'],                    event['component'],
event['resource'],event['timestamp'])

 if event['source_type'] == "resource":
    routing_key += ".%s" % event['resource']

 # Connection
 with Connection(hostname=host, userid=user, virtual_host=vhost) as conn:
    # Get one producer
    with producers[conn].acquire(block=True) as producer:
        # Publish
        producer.publish(
            event,
            serializer='json',
            exchange=exchange,
            routing_key=routing_key)


 sys.exit(0)
```

Cela permet un rendu visuel filtrable des alertes émises sur *Canopsis*, par exemple avec le *widget List*, ainsi qu'on peu l'observer sur l'Illustration 4.6 .

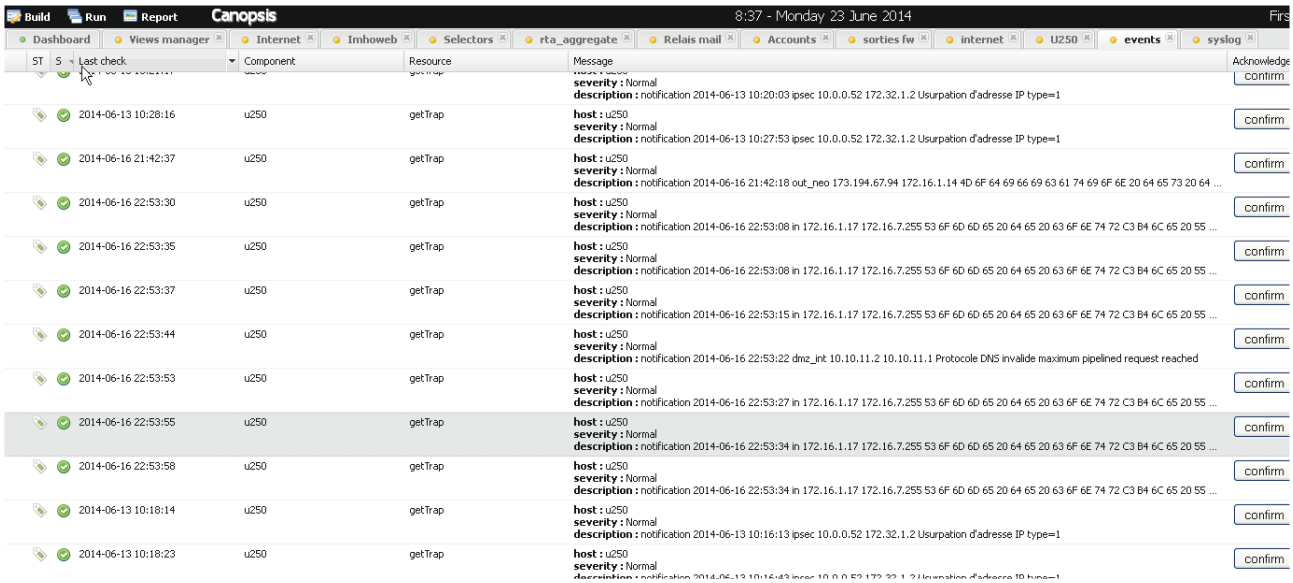


Illustration 4.6 : rendu des traps SNMP via le widget Canopsis « List »

4.2.4. Intégration des messages *Syslog*

La majorité des composants du SI sont capables de générer des messages suivant le protocole *Syslog*[62] défini par la RFC 5424[121]. Au même titre que pour les alertes SNMP passives, il peut être pertinent de mettre en lien des métriques de performance avec des événements émis par un système d'exploitation.

La logique du connecteur d'événements *Syslog* est très semblable à celle que l'on utilise avec les *traps* SNMP. La différence tient au fait que l'on exploite les facultés de redirection d'un *daemon Syslog-ng*[63] central qui redirige les événements externes vers un programme transformant le message *Syslog* en événement *Canopsis*.

Ci-dessous figure un extrait de la configuration *Syslog-ng* qui redirige ses sorties vers un programme nommé *log2canopsis.py* :

```
 log { source(s_src); filter(f_crit); destination(d_console); };
 source ext {
   udp();
   tcp();
   };
 destination to_canopsis {
    program("/usr/bin/python -u   /var/lib/shinken/libexec/log2canopsis.py"      template("$PROGRAM
$HOST $MSG\n")       template_escape(no)
   flags(no_multi_line)
   flush_lines(1)
   flush_timeout(1000)
           );
   };
 log {
   source(ext);
   destination(to_canopsis);
   };
```

La logique du programme de transformation *log2canopsis* est très similaire à celle que l'on a pu observer dans le paragraphe précédent, à la différence que le type d'événement correspond à la valeur *« log »* :

```
 def sendsyslogevent(prog,hostfrom,message):

     host = "172.16.1.47"
     port = 5672
     user = "guest"
     password = "guest"
     vhost = "canopsis"
     exchange = "canopsis.events"

     event = {
      "timestamp":        int(time.time()),
      "connector":        "syslog_trap",
      "connector_name":   "syslog",
      "event_type":       "log",
      "source_type":      "syslog",
      "component":        str(hostfrom),
```

62 http://fr.wikipedia.org/wiki/Syslog

63 Syslog-ng est une implémentation d'un *daemon* syslog. À ce propos de Syslog-ng, voir ; http://www.balabit.com/fr/network-security/syslog-ng/opensource-logging-system

```
        "resource":             str(prog),
        "state":        0,
        "state_type":       1,
        "output":               str(message),
        "display_name":     "SYSLOG"
        }

    routing_key   =   "%s.%s.%s.%s.%s.%s.%s"   %   (event['connector'],   event['connector_name'],
event['event_type'], event['source_type'], event['component'],event['resource'],event['timestamp'])
     exchange = "canopsis.events"
     with Connection(hostname="172.16.1.47", userid="guest", virtual_host="canopsis") as conn:
         with producers[conn].acquire(block=True) as producer:
             producer.publish(
                 event,
                 serializer='json',
                 exchange=exchange,
                 routing_key=routing_key
                 )
     return True
```

À condition de pouvoir sélectionner les événements *Syslog* pertinents, on peut mettre en corrélation les données de performance avec des événements *Syslog* dans *Canopsis*. C'est ce que représente l'Illustration 4.7.

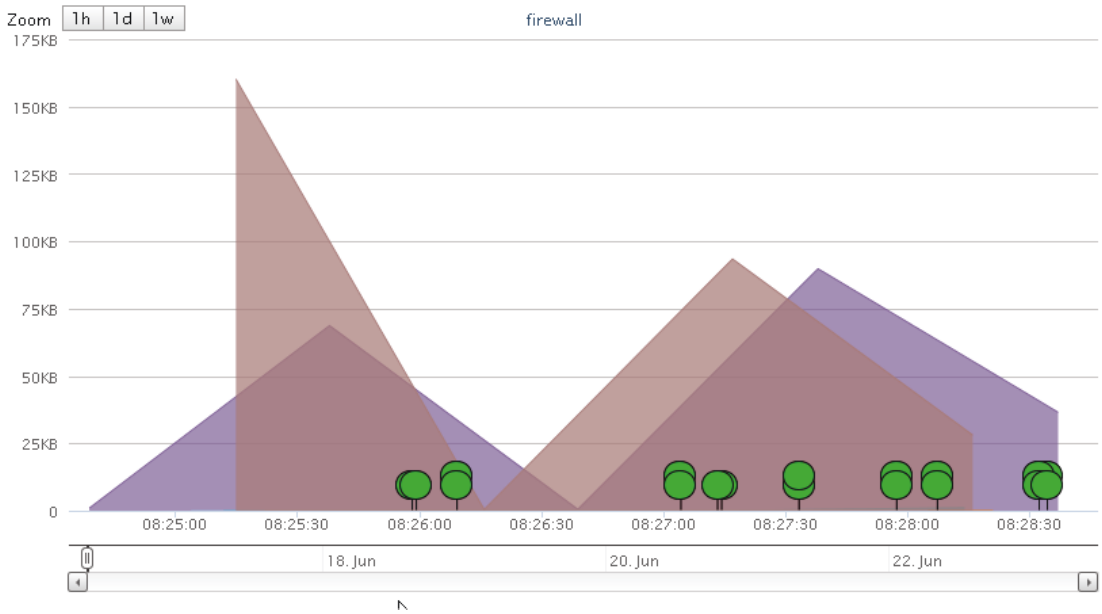


Illustration 4.7 : corrélation entre les événements syslog et les variables de performance d'un firewall netasq U250

4.3. Indicateurs complexes

4.3.1. Les *triggers Shinken*

La logique des *triggers* - ou déclencheurs - a été évoquée au paragraphe 3.1.2.3. Dans le cadre de ce projet, peu de ces *triggers* ont été mis en œuvre. D'une part la construction d'indicateurs composés est réalisable par *Canopsis* et d'autre part, les *triggers* sont des inclusions de code Python qui peuvent, lorsqu'il induisent des traitements trop coûteux en ressource, mettre en péril la charge du serveur *Shinken*.

L'intérêt de l'utilisation des déclencheurs, dans notre cas particulier, a consisté à pallier certaines insuffisances dans les données de performance et plus spécifiquement sur des plug-ins dont le code est soumis à des corrections fréquentes de la part de leurs auteurs. De cette façon, les *triggers* permettent d'isoler un code correctif.

Par exemple, le plug-in *check_vmware_api* ne fournit qu'un pourcentage de l'espace occupé par des disques virtuels sur un système de fichiers *vmfs*[64]. Pourtant, le volume exact en mégaoctets constituait une mesure nécessaire à l'évaluation du projet de migration d'un SAN. Pour éviter de corriger directement le code du *plug-in*, qui est soumis à de fréquentes révisions, nous avons préféré isoler notre correction spécifique dans un *trigger*, de façon à la rendre indépendante du code d'origine. Le code de ce *trigger* consiste simplement à récupérer les valeurs des données de performance et à transformer le pourcentage d'espace occupé en un volume précis :

```
 import re
 myobj=get_object('ventouse/Vmfs Checks')
 #output=self.output
 mydatas=myobj.output
 datas=str(mydatas).split(',')
 perf=""
 for data in datas:
     pattern = re.compile(r"\'(.*.)\'.used.\=(.*.)\sMB*.\((\d+\.\d+)%")
     match=pattern.findall(data)
     if match:
         dsname=match[0][0]
         dsused=float(match[0][1])
         pc=float(match[0][2])
         totalcapa=round(float(dsused * 100 / pc),2)
         volwarn=round(float(totalcapa*95/100),2)
         volcrit=round(float(totalcapa*98/100),2)
         perf=perf+"'"+str(dsname)+"'="+str(dsused)+"MB;"+str(volwarn)
+";"+str(volcrit)";0;"+str(totalcapa)+"; '"+str(dsname)+"_pc'="+str(pc)+"%;95;98;0;100 "
         if (float(pc)>96):
             exit_code=2
         else :
             exit_code=0
             myoutput=str(myobj.output).replace('CRITICAL','OK')

 set_value(myobj, output=myoutput,perfdata=perf,return_code=exit_code) Programming Interface
```

64 Vmfs : V*irtual Machine File System* est un système de fichier propre à l'infrastructure de virtualisation VMware. Cette dernière est destinée à contenir des disques durs virtuels utilisés par les machines virtuelles.

Comme on le constate, on peut remplacer assez facilement les données de performance et le code de sortie du *plug-in*. Dans la même idée, on peut construire de façon programmatique des KPI en effectuant des calculs sur les données de performance. L'exemple ci-dessous montre comment obtenir un KPI représentant les temps de réponse moyens de requêtes HTTP :

```
print "TRIG: I am a trigger in the element", self.get_full_name()
names = ['srv-web-%d/Http' %i for i in range(1, 4)]
srvs = [get_object(name) for name in names]

print "TRIG: Got http services", srvs
perfs = [perf(srv, 'time') for srv in srvs]
print "TRIG: Got perfs", perfs
value = sum(perfs, 0.0)/len(perfs)
print "TRIG: and got the average value", value

print "Now saving data"
self.output = 'Trigger launch OK'
self.perf_data = 'HttpAverage=%.3f' % value
```

Ce type de calcul a néanmoins l'inconvénient de reposer sur le nommage littéral des objets supervisés : on suppose ici que tous les hôtes HTTP sont désignés par la chaîne de caractères 'srv-web-%d' dans laquelle %d représente un nombre décimal. De fait, à moins qu'une nomenclature de nommage stricte ne soit établie, ce code peut perdre en lisibilité.

Un autre exemple a été illustré au paragraphe 3.1.2.3. Il concerne la construction d'un compteur du nombre total de pages imprimées sur l'ensemble du SI. Le code du *trigger* permet de rendre cohérente l'évolution de la valeur du compteur et de visualiser la répartition des impressions sur l'ensemble du SI. Sans cette consolidation, l'indicateur retombe à zéro en même temps que les imprimantes, faussant ainsi le résultat. À titre d'exemple, l'Illustration 4.8 montre le rendu graphique de cet indicateur dans l'interface *Canopsis*.

Illustration 4.8 : exemple de rendu de KPI sur les compteurs d'impression dans Canopsis

Encore une fois, l'utilisation de *triggers Shinken* relève davantage d'une action corrective à posteriori que de la construction d'un KPI, ce rôle étant déféré à *Canopsis*. Cet usage s'explique notamment par les facilités ergonomiques de l'interface *Canopsis* et les besoins d'ajustement des KPI au moment de leur élaboration.

4.3.2. Construction d'indicateurs dans *Canopsis*

Les moteurs *Canopsis* autorisent le calcul de l'état global d'une série d'événements. La technique consiste à assembler des événements en un indicateur composite et à affecter à ce dernier l'état calculé de l'ensemble de ces sous-états. On peut par exemple appliquer à l'ensemble de ces composants l'algorithme *worst state* : l'ensemble des composants est défaillant dès lors qu'un de ses sous-composants l'est. On peut également créer des liaisons logiques de toutes pièces comme avec le *widget topology* dont on observera un exemple plus loin. Le *widget wheather* permet, quant à lui, de présenter de façon simplifiée l'état d'un groupe logique de composants.

Tous les indicateurs de performances reçus dans *Canopsis* peuvent être transformés par le moteur de « consolidation ». *Canopsis* y distingue la notion *d'agrégation* et celle de *consolidation* : la première concerne l'intervalle de temps durant lequel un moteur *Canopsis* va utiliser les données récoltées. La consolidation concerne le type d'opération que le moteur va mettre en œuvre sur l'ensemble des données récoltées durant l'intervalle d'agrégation. Il s'agit d'opérations arithmétiques simples : maximum, minimum, moyenne, différence, etc. Les possibilités de raffinement sont moindres que dans *Shinken*, mais possèdent l'avantage de pouvoir être lisibles par n'importe quel utilisateur.

Illustration 4.9 : Canopsis - vue « consolidation »

4.3.2.1. Selectors et SLA

Les sélecteurs *Canopsis (Selectors)* désignent un *méta-objet* qui restitue l'état global d'un lot d'autres objets. Il s'agit de déterminer un *méta-état* - soit un état d'état - à partir d'une série de résultats de tests. Le calcul d'état se fait simplement à l'aide d'une boîte de dialogue (Illustration 4.10). Les moteurs *Canopsis* évoqués plus haut sont ensuite en mesure de produire des calculs d'états à partir de ces sélecteurs.

Illustration 4.10: construction d'un selector et d'un SLA associés sous Canopsis

Le SLA associé peut être calculé automatiquement, par exemple en vérifiant la proportion de temps durant laquelle le *selector* a été en état valide *(OK)* sur une période. Cette recomposition permet de simplifier l'évaluation de l'état de composants complexes et de leur associer un SLA, sans être contraint par l'articulation des sondes techniques avec des hôtes et des services. La possibilité offerte par la logique *Nagios* de dissocier les objets permet de pointer des sous-ensembles d'un hôte, ou d'assembler des résultats de test provenant de divers hôtes à la volée. La sélection du lot d'éléments se fait aussi à l'aide d'une boite de dialogue que l'on retrouve dans l'Illustration 4.11.

Illustration 4.11: onglet « filtre » - définition d'un sélecteur Canopsis

Une fois construits, les SLA sont accessibles et modifiables à partir de l'interface d'administration *Canopsis*. C'est ce que représente l'Illustration 4.12.

Illustration 4.12 : liste des selectors et des SLA associés dans l'interface Canopsis

Les SLA s'obtiennent donc à partir d'un calcul interne à *Canopsis*. Il est ensuite possible de valoriser ces indicateurs au travers d'une vue : c'est ce que l'on retrouve dans l'Illustration 4.13.

Le calcul des SLA se paramètre donc exclusivement via l'interface graphique. Malgré les facilités que cela procure, les possibilités d'affiner les sélecteurs demeurent limitées à la structuration des messages envoyés à *Canopsis*. C'est ainsi qu'une inversion des valeurs des composants et ressources[65] dans le module *Canopsis* de *Shinken* a compromis la base de données *MongoDB* de *Canopsis* durant les phases de récolte des données. Une autre difficulté tient à l'impossibilité d'utiliser des conditions logiques semblables à celles que l'on retrouve dans le moteur *topology*.

65 https://github.com/shinken-monitoring/mod-canopsis/issues/4

Illustration 4.13 : représentation de SLA dans Canopsis

Une fois que l'exécution des sondes *Shinken* est tempérée par les horaires de travail, le rendu des SLA témoigne de la *disponibilité* d'un ou plusieurs composants sur une période de temps donnée. Le calcul de SLA constitue une analyse à posteriori et sa valeur ne prend sens que sur la durée. On peut toutefois utiliser directement les valeurs des sélecteurs pour s'approcher au plus près de l'état actuel du système. C'est typiquement ce que l'on fait avec les *widgets* de type *weather*, en construisant un indicateur complexe valide sur des périodes de temps plus courtes.

4.3.2.2. Représentation graphique d'un sélecteur

La mise en place d'un sélecteur renvoie donc un *méta-état.* Ce dernier, au même titre qu'une sonde *Nagios*, est assorti de quatre valeurs possibles : *OK* (0), *Warning* (1), *Critical* (2) et *Unknown* (3). L'intérêt principal des sélecteurs est de synthétiser l'état d'architectures complexes. Graphiquement, on peut utiliser le *widget Weather* qui suggère le résultat à l'aide d'une iconographie météorologique :

Illustration 4.14 : exemple d'utilisation du widget Weather

L'intérêt de ce type de *widget* est de synthétiser les informations de supervision en temps réel. En ce sens, ils sont davantage adaptés aux utilisateurs profanes, soucieux de connaître l'état actuel des services que procure le SI. Le cas échéant, il est possible d'adjoindre un lien vers une vue technique plus détaillée. En dissociant la représentation des opérations d'agrégation et de calcul, on masque la complexité sous-jacente de l'infrastructure du SI mais en contrepartie, on facilite sa visibilité.

4.3.2.3. Topology

Le *widget topology* permet d'établir un *méta-état* à partir du chaînage logique d'un ensemble de composants. On cherche par exemple à évaluer le fonctionnement du DNS pour l'ensemble du SI, sachant que ce dernier est organisé de façon spécifique. Ce cas d'espèce se présente de la façon suivante : trois contrôleurs de domaine assurent la résolution de nom dans le LAN. Lorsqu'une requête doit être adressée à un DNS externe, elle transite par le *firewall* qui n'autorise que deux DNS externes. Le *firewall* propose donc un service DNS dont on ne peut se passer pour la résolution externe. L'interface *Canopsis* permet de restituer un indicateur capable de dire si le service DNS est entièrement opérationnel. Il faut pour cela :

- établir ce à quoi correspond la notion de *disponibilité de DNS* : nous nous appuyons pour cela sur des tests issus de *Shinken* vérifiant le temps de réponse d'une requête DNS. Dans le cas des serveurs *Active Directory*, on adjoint un test WMI ;
- que l'un des trois DNS internes soit disponible ;
- que le service DNS du *firewall* soit disponible ;
- que l'un des deux DNS externes autorisés soit disponible.

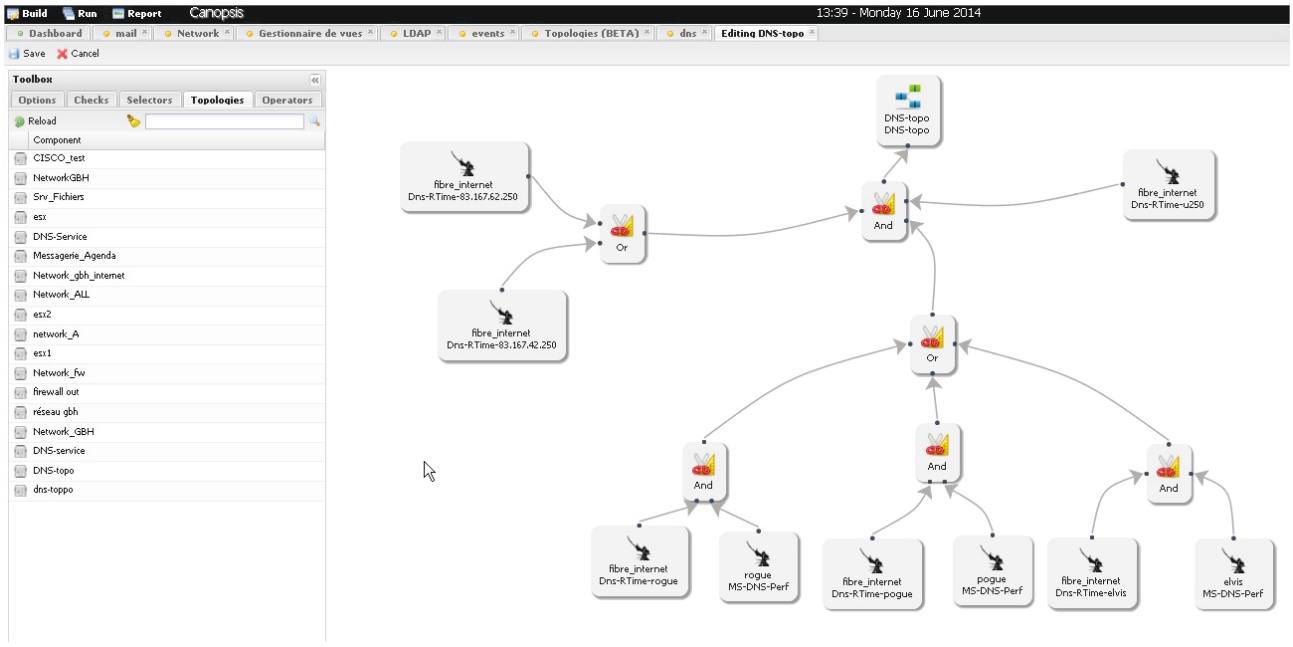


Illustration 4.15 : exemple de construction d'une topologie Canopsis

On traduit cette suite de conditions sous forme graphique : c'est ce que montre l'Illustration 4.15. Une fois la topologie construite, *Canopsis* calcule l'état global du service DNS, en indiquant éventuellement l'élément défaillant (voir l'Illustration 4.16).

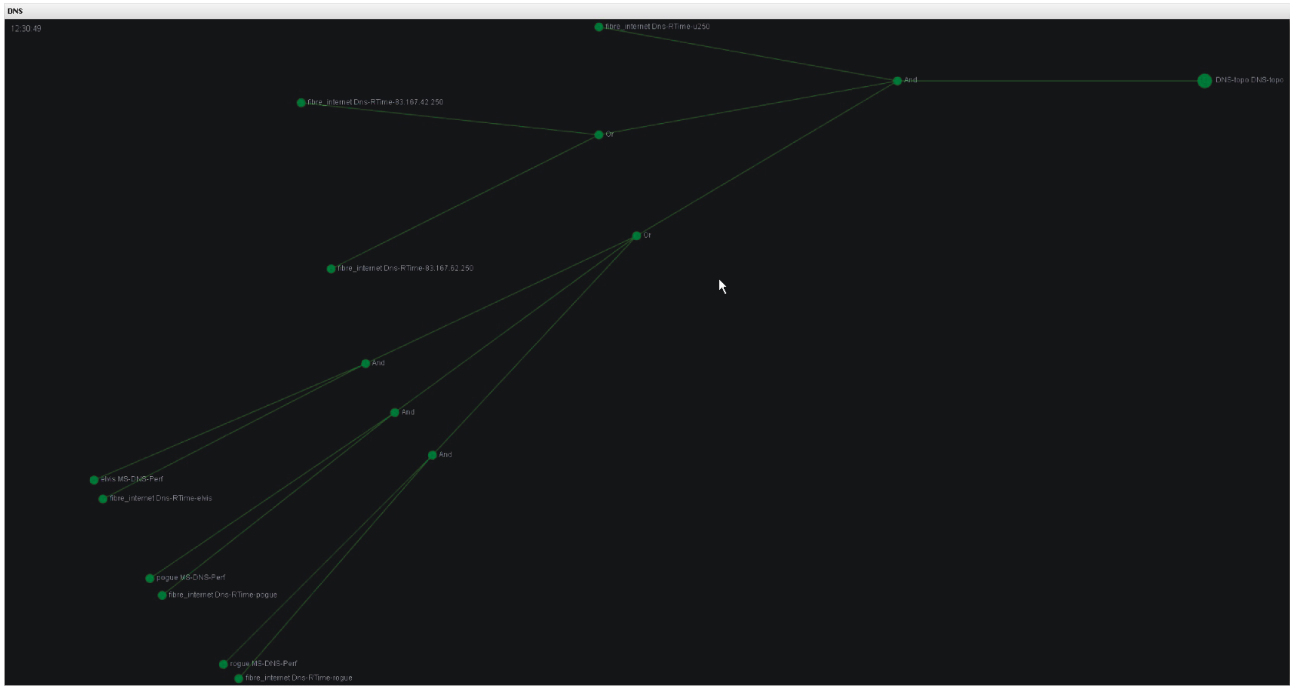


Illustration 4.16 : rendu graphique du widget topology

De cette façon, il est possible d'associer hôtes, résultats de tests, sélecteurs ou topologies *Canopsis* dans un même graphique. Selon les conditions spécifiées dans un nœud *(and, or, cluster, worst)*, l'état *(OK, Warning, Critical)* se répercute sur ses parents.

Illustration 4.17 : propagation d'un « warning » sur les noeuds parents d'un widget topology

4.4. Évolutions prévues

Les vues ont été organisées en fonction des différentes entités identifiées lors de l'analyse : stockage, réseau, virtualisation, plates-formes, systèmes, applications, etc. Des extraits de ces vues figurent en annexe de ce mémoire dans la partie 6.9.

L'infrastructure *Shinken* mise en œuvre à GBH dénombre 83 *hôtes* et plus de 1400 *services*. Bien que quelques hôtes ne soient pas encore intégrés dans les processus de supervision, cette charge représente un faible volume au vu des capacités théoriques de *Shinken*. Rétrospectivement, la véritable difficulté a sans doute été d'ajuster les intervalles entre les différents *checks* pour éviter toute surcharge système, tout en autorisant un niveau de précision suffisamment élevé sur la collecte de certaines mesures (réseau, E/S disques par exemple).

La totalité des résultats des tests *Shinken* est transférée au serveur *Canopsis,* où l'on compte une trentaine de vues exploitables. Les s*électeurs* peuvent regrouper une centaine de tests chacun et sont également déclinés sous forme de *topologie*, de façon à expliciter les modalités d'évaluation. Il reste néanmoins un certain nombre de vues à affiner ou à créer.

Actuellement, les possibilités de mise à disposition des vues *Canopsis*, par export ou par l'accès web, remportent assez peu de succès auprès des utilisateurs. Cet état de fait s'explique par le caractère expérimental du projet. L'intérêt pour ce type de travail, s'il semble évident pour un service informatique, demeure beaucoup plus obscur pour les utilisateurs et les décideurs. Non qu'il s'agisse d'un échec, mais il semble que la diffusion des informations et la sensibilisation des utilisateurs consistent en un besoin d'organisation à part entière. Le projet a montré d'un côté la faisabilité et les limites des solutions de supervision et d'*hypervision* et de l'autre, la nécessité d'impliquer davantage les utilisateurs dans l'évaluation du système d'information.

Si la mise en place des plates-formes est stabilisée, il reste néanmoins quelques évolutions à prévoir :

- les premières sont inhérentes à l'évolution des projets *Shinken* et *Canopsis*, dont le développement est particulièrement actif. La dernière version « stable » de *Canopsis* date toutefois de décembre 2013 et les versions dites « de développement », malgré quelques tests, demeurent difficilement utilisables dans un environnement de production. Un certain nombre d'améliorations sont attendues sur la prochaine version stable : elles portent notamment sur l'interface graphique, la documentation et les facilités de gestion de la base *MongoDB*. *Shinken*, quant à lui, en est actuellement à sa version 2.0.3. Le développement des *packs* et des *modules* est dissocié du dépôt *git* de *Shinken* mais témoigne également d'un développement

actif[66]. Cette activité nécessite une veille et la prévision de mises à niveau ;

- les secondes évolutions portent sur des adaptations propres au SI de GBH : si les communautés sont actives, il reste rare que les codes source existent ou soient parfaitement adaptés. Ainsi, un certain nombre de sondes nécessitent des corrections ou améliorations. Du côté des sondes techniques, les quelques véritables difficultés identifiées sont très souvent liées à la présence de licences logicielles empêchant l'automatisation. Il s'agit notamment de :
    - l'acquisition des données de performance des SANS EMC VNX5100 : seul un client graphique sous licence peut y accéder,
    - l'accès aux informations des serveurs Citrix : le service SNMP est soumis à une licence dont on ne dispose pas,
    - l'accès aux MIB des copieurs Canon qui oblige à une interprétation limitée des OIDs SNMP,
    - l'accès aux informations de transfert de l'EDI *use it flow*[67], là encore soumis à licence,
    - l'accès aux moteurs d'impression de l'ERP *Aravis* dont le code est inaccessible.

    Bien que des solutions de contournement[68] soient parfois envisageables, elles demeurent rarement satisfaisantes.

Du côté des sondes comportementales, les tests menés à l'aide de *Sikuli* demeurent difficiles à stabiliser, du fait de perturbations entravant le procédé de reconnaissance optique. Hormis en dédiant une station de travail à l'usage de ce type de tests, aucune solution n'est envisageable aujourd'hui. Par ailleurs, si la difficulté de piloter les tests à distance a été surmontée, il reste à affiner et à varier les scénarios de tests. En certains cas, l'utilisation de *Sikuli* peut être envisagée comme un moyen de contournement des difficultés précitées, tout en gardant à l'esprit que le procédé reste fragile : un mauvais ordonnancement de tests comportementaux interfère avec le bon déroulement de ceux-ci. En ce qui concerne ces tests EUE, il reste donc un travail à mener avec les services concernés, pour mieux évaluer la pertinence des scénarios de tests et en élaborer de nouveaux. À ce niveau, il pourrait être pertinent d'utiliser des outils tels que *Cucumber[122]*, qui introduit la possibilité d'utiliser un langage naturel[123] pour l'élaboration

66 Voir notamment : https://github.com/naparuba/shinken/graphs/

67 http://www.prologue.fr/pages/flux-habitat-social/

68 Il est toujours envisageable d'automatiser des procédés pour récupérer des informations : *Sikuli* en est un exemple. Pour *Citrix*, on parvient toutefois à obtenir quelques information sur le taux d'utilisation des licences par le protocole WMI. Le risque d'alourdir la complexité programmatique et la latence des sondes demeure néanmoins important.

de scénarios comportementaux. Cette pratique est d'ailleurs encouragée par la méthode *Behiavour Driven Development* qui encourage la collaboration entre les techniciens et les utilisateurs[124].

Par ailleurs, lorsque cela est techniquement et légalement possible, il serait envisageable d'améliorer le suivi de certains processus métier, non pas en simulant un comportement, mais en introduisant des sondes de façon intrusive au sein même des programmes. Il s'agit d'une des techniques répertoriées par le modèle APM (voir l'Illustration 2.16) À titre d'exemple, des sondes ont été introduites dans les scripts de transferts automatisés entre l'ERP *Aravis* et une plate-forme distante *imhoweb* qui gère le processus d'attribution des demandes de logement social. C'est ce que montre l'Illustration 4.18.

Illustration 4.18 : graphique produit à partir de sondes applicatives intrusives

On pourrait également envisager d'étendre l'usage des données récoltées à l'amélioration du calcul de tendances. Actuellement, *Canopsis* sait générer des courbes de tendances *(trend)*, mais uniquement en appliquant des régressions linéaires simples. Le procédé est donc totalement inefficient, par exemple dans le cas de séries temporelles faisant apparaître des saisonnalités. Pour y remédier, il faudrait sans doute mettre en œuvre d'autres outils spécialisés dans l'analyse statistique, comme *R*[69]. Pour autant, ce type de traitement relève davantage de calculs statistiques et reste délicat à intégrer[70].

69 *« R »* est un langage de traitement de données et d'analyse statistique : http://www.r-project.org/

70 Sur de plus anciennes versions de *Shinken*, l'expérience d'un module dédié au calcul de tendances permettait de lancer des calculs de courbes de tendance à intervalles réguliers. Il reste très peu documenté, sans doute en raison des difficultés à généraliser et à automatiser ce type de traitement.

5. Conclusion

Lors de la mise en place de ce projet, nous avons d'abord travaillé sur une approche analytique du SI. La librairie ITIL fournit à ce niveau un premier ensemble de définitions génériques qui rappellent le lien entre l'activité du SI et celle de l'organisation. Partant de cette idée d'alignement stratégique, l'analyse des fonctions du SI met à jour deux notions importantes :

- d'un côté, le SI détient des objectifs fonctionnels qui lui sont propres : sécurité des accès, sauvegarde des données, performance des applications, intégration des solutions, etc. Ces axes sont notamment décrits par le modèle ITIL ;

- d'un autre côté, le SI doit satisfaire aux besoins de l'organisation et vérifier que les services qu'il fournit sont en adéquation avec l'activité de cette dernière.

Cette double contrainte explique que l'on distingue :

- un aspect *opératoire*, qui est typiquement destiné à l'usage des gestionnaires du SI. La solution *Shinken* correspond davantage à cet usage ;
- un aspect plus *analytique*, qui correspondrait davantage à des *tableaux de bord décisionnels*. *Canopsis* prend en charge cet aspect en induisant des analyses à posteriori, éventuellement désynchronisées de l'activité de production en temps réel. À ce niveau, on tente de rendre *visible* le SI et de mesurer son *alignement stratégique*. C'est ce que l'on fait avec les tests comportementaux, les KPI et SLA.

Ces exigences sont largement motivées par les préconisations ITIL. Celles-ci rappellent que l'activité d'un service informatique ne se construit pas ex-nihilo, mais constitue bien une réponse adaptée à un besoin. La supervision ainsi mise en place s'oriente autour de quatre notions : disponibilité, accessibilité, performance et conformité. De façon très synthétique, on a pu observer ces trois points :

- les SLA sont fondés sur l'idée d'un contrat qui positionne le SI comme un fournisseur. Ils donnent une indication de *disponibilité ;*
- les sondes techniques procurent des indications de *performance* ou vérifient *l'accessibilité* d'un service à un moment donné ;
- les tests comportementaux, quand à eux évaluent la *conformité* du service en reproduisant des scénarios.

Les plates-formes de supervision et d'*hypervision* agissent en complémentarité dans l'objectif de restituer l'activité du SI, à la fois pour ses gestionnaires et pour les utilisateurs.

En pratique, la plate-forme *Shinken* s'est révélée particulièrement efficace dans le domaine de la détection d'incidents et dans celui des diagnostics : les coupures électriques et les défaillances d'un composant logiciel sont correctement détectées et remontées. *Canopsis* a permis de restituer de façon synthétique les informations remontées, voire même d'agréger d'autres informations comme les journaux *Syslog*, certaines sondes applicatives, ou bien des *traps* SNMP.

Ainsi que nous l'avons observé, le concept *d'hypervision* tel que *Canopsis* le définit vise à unifier différentes techniques de supervision, en transposant les notions d'événements, de données de performance, de composants et de ressources en des concepts englobants suffisamment vagues pour s'adapter à différents contextes.

Au-delà des différentes possibilités graphiques et esthétiques offertes par *Canopsis*, il a semblé pertinent d'étendre le champ d'action de la supervision à celui de *l'hypervision* pour deux raisons :

- d'une part, le champ de la supervision, bien qu'il puisse s'étendre à souhait, est contraint par une exigence opérationnelle immédiate. La sélection des informations pertinentes nécessite une analyse préalable, mais également un examen à posteriori, par exemple pour vérifier la pertinence et la précision des sondes mises en œuvre. Concrètement, la vérification systématique de tous les journaux système par l'intermédiaire de sondes génère davantage de bruit que d'éléments de diagnostic pertinents. En insistant sur les représentations graphiques et leur sémantique, *Canopsis* œuvre davantage comme un outil de restitution et d'analyse complémentaire à *Shinken* ;
- d'autre part, *Canopsis* permet de *surcharger* le modèle de supervision *Nagios* dont hérite *Shinken*. Or, ce modèle est particulièrement marqué par le point de vue des gestionnaires de l'infrastructure : on y énumère des hôtes dotés de services, que l'on regroupe éventuellement en *services groups* ou *hosts groups,* mais sans qu'émerge la finalité de l'infrastructure. Cet état de fait, qui est également un héritage, témoigne d'une lacune dans cette approche : l'utilisateur n'y est pas évoqué et l'infrastructure semble exister pour elle-même. Les *business rules* de *Shinken* permettent de surpasser cette lacune et rappellent, à l'instar d'ITIL, que l'objectif de l'infrastructure consiste bien en la satisfaction de ses utilisateurs finaux. *Canopsis* met en exergue cet aspect en favorisant d'un côté un ré-assemblage des dépendances - par exemple avec les topologies - et de l'autre, en permettant l'élaboration interactive d'indicateurs agrégés.

Ce passage de la supervision à l'*hypervision* est intéressant à plusieurs égards :

- il rappelle que le SI n'est pas uniquement un lot de composants techniques : c'est une construction dont on peut évaluer l'efficacité en confrontant l'outil à ses

usages, notamment en s'appuyant sur des tests comportementaux et en accompagnant les résultats d'une restitution graphique. N'importe quel utilisateur peut accéder à des informations de supervision dont dépend son activité : mails, accès internet, plates-formes externes, liaisons internes. La question de la sémantique des représentations n'en est pas pour autant résolue ;

- par ailleurs, il met en avant le fait que la supervision strictement technique n'offre qu'une vision limitative du SI : la transformation d'indicateurs et leur rendu graphique doivent répondre à des exigences de lisibilité pour être accessibles aux usagers. En fin de compte, cela suggère que l'évaluation de l'adaptation du SI aux besoins de l'activité métier pourrait également être envisagée de façon active par ses utilisateurs. Cette exigence de lisibilité ne vise pas l'exhaustivité du SI, mais cherche davantage à *valoriser l'interaction entre le SI et les utilisateurs ;*
- enfin, l'évaluation d'un SI ne consiste pas uniquement en celle de son activité opératoire immédiate. Elle participe à organiser et à prévoir son évolution. À titre d'exemple, l'acquisition d'un nouveau SAN fibre optique au cours de l'année 2014 repose sur l'analyse des données de supervision : espace occupé, répartition de I/O, taux de lecture sont obtenus grâce à l'historisation de données de supervision et aux projections que l'on en déduit.

Si le SI est construit comme un ensemble cohérent de composants, les représentations que l'on obtient ne procèdent pas systématiquement des mêmes points de vue. Ceci explique que sa sémantique varie selon que l'on se place du point de vue de l'utilisateur ou de celui de l'infrastructure. Ainsi qu'on a pu le remarquer, les gestionnaires du SI possèdent leurs propres objectifs fonctionnels. Non qu'il s'opposent à ceux de l'organisation, mais leurs points communs demeurent parfois difficiles à représenter. Certains produits valorisent en effet très exclusivement les techniques dites d'APM[54], sans doute au détriment du point de vue des gestionnaires du SI. Pour éviter cette tentation antagonique, certaines méthodes, comme celles reposant sur le langage naturel[124] [123] peuvent constituer des pistes de réflexion pertinentes. Si cela témoigne d'une intention de (ré-)conciliation des deux points de vue, les effets n'en sont pas flagrants. Cela s'explique par deux raisons :

- d'une part, les questions de supervision ont été historiquement le fait de techniciens qui ont, avant tout, valorisé le point de vue de l'infrastructure. Bien que dénotant une certaine efficacité, cette approche contribue à valoriser la complexité technique du système et détériore conséquemment sa lisibilité pour les profanes.
- d'autre part, de cette même complexité émane une forme de déni[125], dont les effets tendent à réduire le SI à une « boîte noire ». Or, ainsi que nous avons pu le constater en mettant en place une infrastructure de supervision, la réelle problé-

matique de la visibilité du SI consiste bien à regrouper une multitude de points de vue. C'est cette complexité, caractérisée par l'impossibilité pour un individu de concevoir seul *une vision globale et exhaustive du système[126]*, que l'on rencontre en tentant d'établir un modèle uniforme du SI en vue de le superviser.

Dans le fond, la mise en œuvre d'un système de supervision ne renvoie pas à une impossibilité technique, mais questionne les modalités de représentation du SI. Par exemple, le modèle *Nagios* s'appuie sur une représentation arborescente du SI. Or, ce mode de représentation n'a de sens que lorsque l'on est en mesure de comprendre les multiples liens d'interdépendances entre les objets techniques supervisés. *Canopsis* permet, dans une certaine mesure, d'isoler des représentations de points de vue différents et d'établir un *compromis.* Les modèles comme CIM ne prennent sens que lorsque le SI est homogène : l'implémentation Microsoft WMI ne peut être utilisée que pour les produits Microsoft. Or, confrontées à un SI hétérogène, ces représentations ne peuvent pas s'appliquer d'emblée. C'est bien ce qui nous a amenés à la recherche d'un compromis, autant en termes d'évaluation technique que de représentation.

Dans la pratique, l'assemblage de *Canopsis* et de *Shinken* permet effectivement d'améliorer la gestion du SI. En ce sens, cet objectif est atteint. Malgré tout, la question de la visibilité du SI reste en suspens : bien que techniquement réalisable, la représentation du SI reste délicate à établir d'emblée - ou du moins, à établir sans le concours de ses usagers. S'il semble évident que les représentations concrétisées par des vues *Canopsis* devraient être élaborées en concertation avec les utilisateurs, ce n'est pas encore le cas. En tout état de cause, un travail reste à faire à ce niveau.

Après une dizaine de mois en production, il convient de noter que l'architecture mise en place démontre des qualités d'adaptation et de stabilité remarquables. Si quelques fragilités ont été relevées, le dynamisme des communautés de développeurs autour de ces projets laisse supposer des évolutions prometteuses.

Au final, ce travail a eu des vertus structurantes sur le SI : le recensement des éléments de l'architecture, l'évaluation des performances, l'aide au diagnostic sont des conséquences directes du processus de supervision. De façon beaucoup moins évidente, ce projet a très largement favorisé une *prise de recul* sur les pratiques et le rôle du service informatique au sein de l'organisation.

En ce sens, on peut dire pour conclure que la supervision ne se limite pas aux notions de contrôle et de surveillance. Elle questionne également le rôle du système d'information. La supervision peut certes être encadrée par des méthodologies, des normes et des préconisations : les *Service Oriented Architectures*, ITIL, le *Business process management* ou encore le *Business Motivation Model* en sont quelques exemples. Ces approches tendent toutefois à idéaliser le système d'information en archétype de la rationa-

lité organisationnelle. L'expérience de la supervision, telle qu'elle a pu être menée au sein de GBH, montre toutefois que cette rationalité ne constitue qu'un point de vue parcellaire. Si l'objectif *affirmé* de l'organisation porte effectivement sur une motivation d'ordre commercial et s'appuie sur des processus rationnels, son fonctionnement *effectif* repose également sur des comportements humains dont les motivations peuvent diverger.

Ce que montre le projet de supervision à GBH, c'est qu'un point de vue méthodique et rationnel ne suffit pas à exprimer la complexité du SI. Il constitue une aide et permet d'isoler un dysfonctionnement, pour autant que des règles aient défini ce que constitue la *normalité :* par exemple, une latence ne peut être considérée comme excessive qu'à partir du moment où des seuils ont été définis. L'idée est donc bien de se reposer sur un système de règles. Pour autant, on ne peut pas éliminer des *zones d'incertitudes où s'expriment des défaillances dans le système de règles de l'organisation[6].* En ce sens, certains modèles peuvent paraître inadaptés, tant ils présupposent l'organisation comme un construit rationnel. À contrario, on pourrait imaginer que la supervision du SI ne s'envisage pas uniquement comme le calcul d'un écart à une norme, mais également comme un moyen de mettre en évidence ces zones d'incertitudes. Ainsi qu'on a pu le constater, il demeure difficile d'établir de façon systématique les contours du SI. On pourrait supposer que cette difficulté soit liée aux limites de ce que les normes, modèles et préconisations admettent comme relevant du champ du système informatique. Les tests EUE présupposent des comportements rationnels, et par là même, ne permettent pas d'expliquer un ressenti ou comportement d'utilisateur qui échapperait à cette rationalité. À ce niveau, un questionnement se rapportant davantage à une optique socio-organisationnelle pourrait constituer un complément pertinent à la supervision du SI.

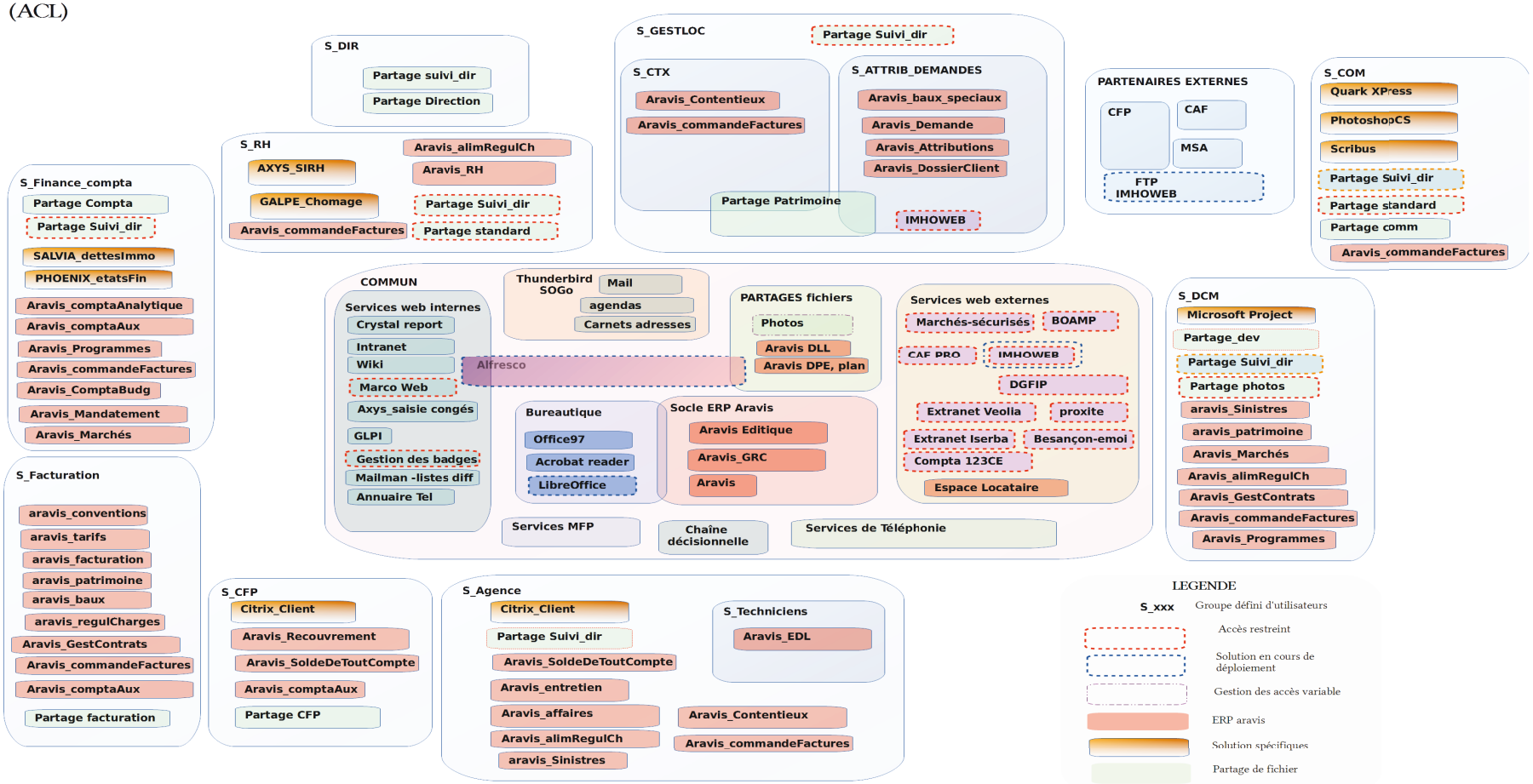
6. Annexes

Table des annexes

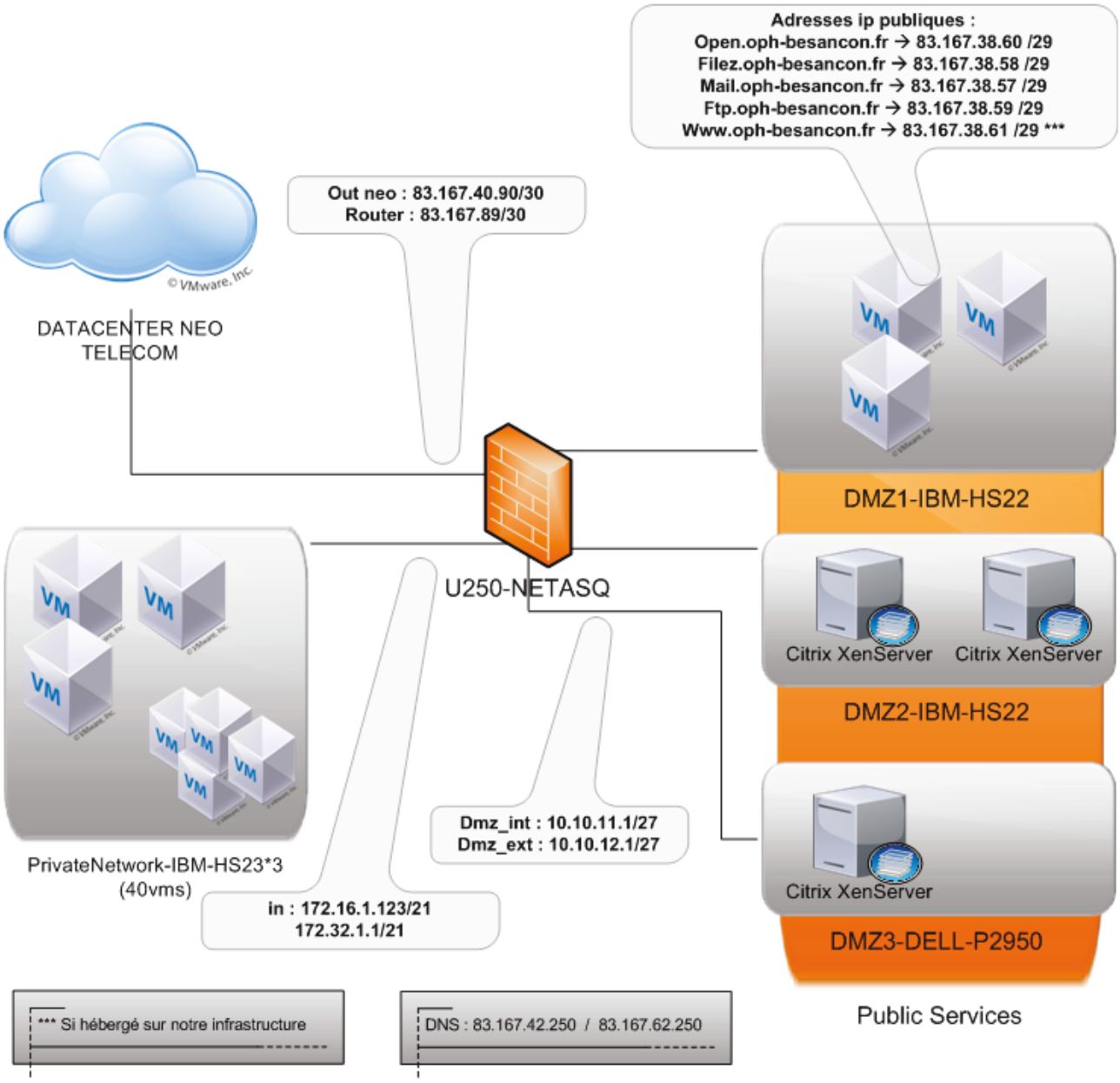
6.1. Organigramme fonctionnel de Grand Besançon Habitat

6.2. Couverture applicative : répartition des accès

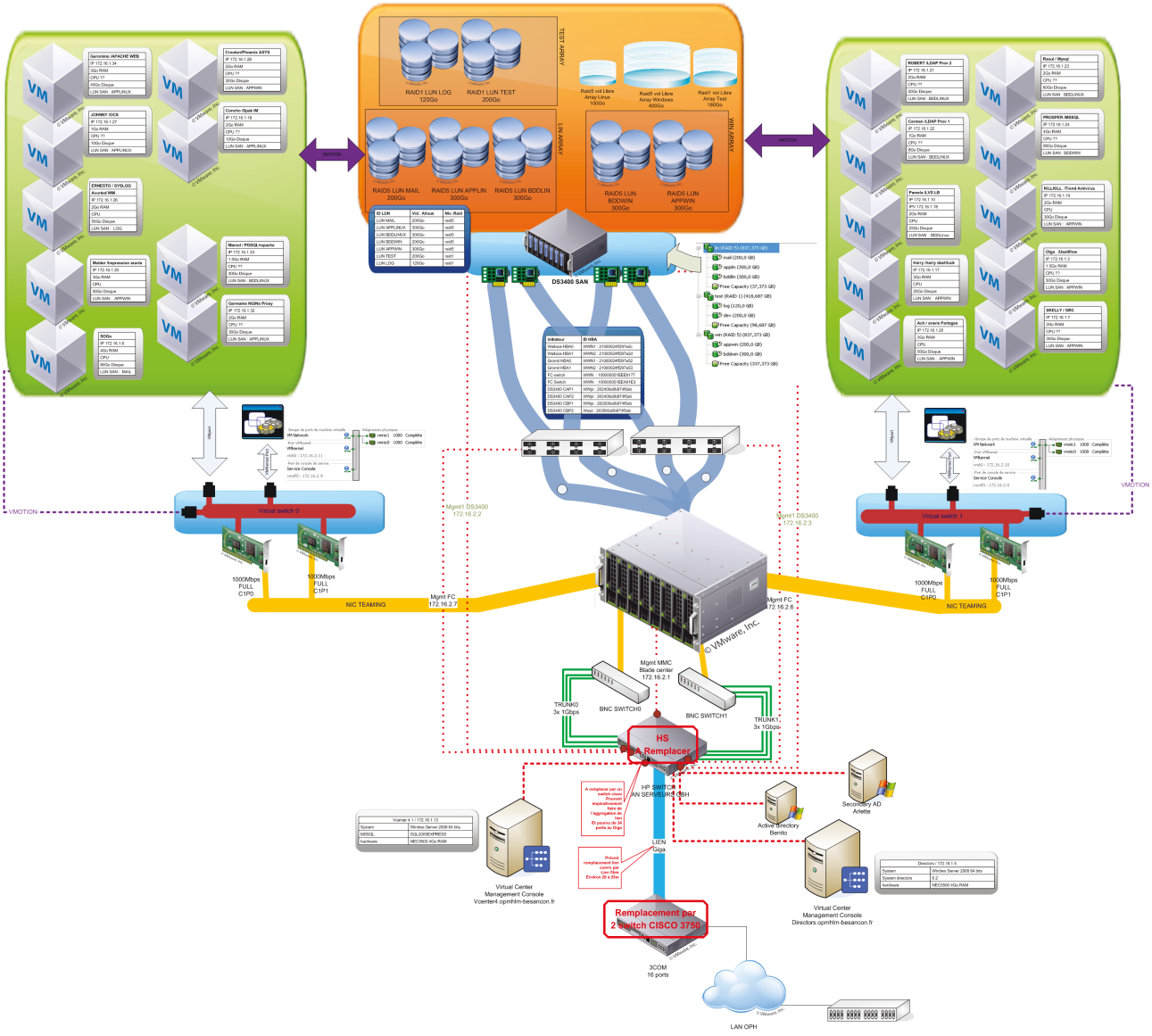
Répartition des accès applicatifs par groupe identifié (ACL)



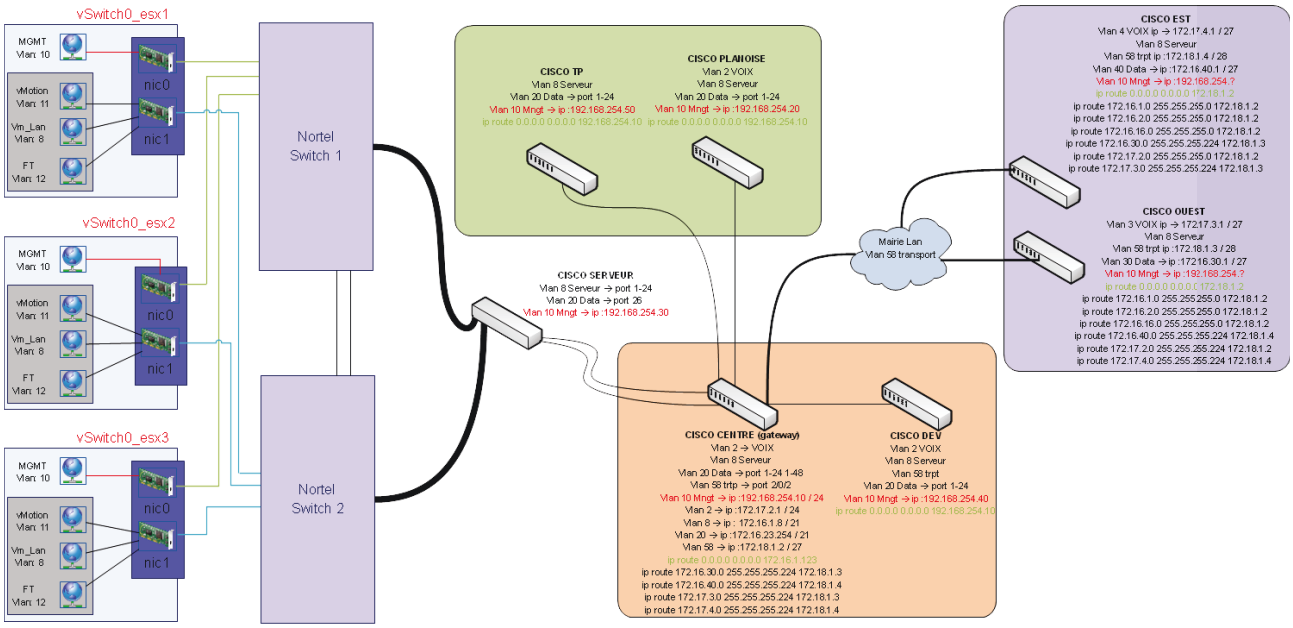
6.3. Schéma de l'architecture DMZ



6.4. Représentation simplifiée de l'architecture virtualisée *ESX* (2010)



6.5. Schéma réseau GBH (2014)



6.6. Processus d'intégration d'un élément à superviser

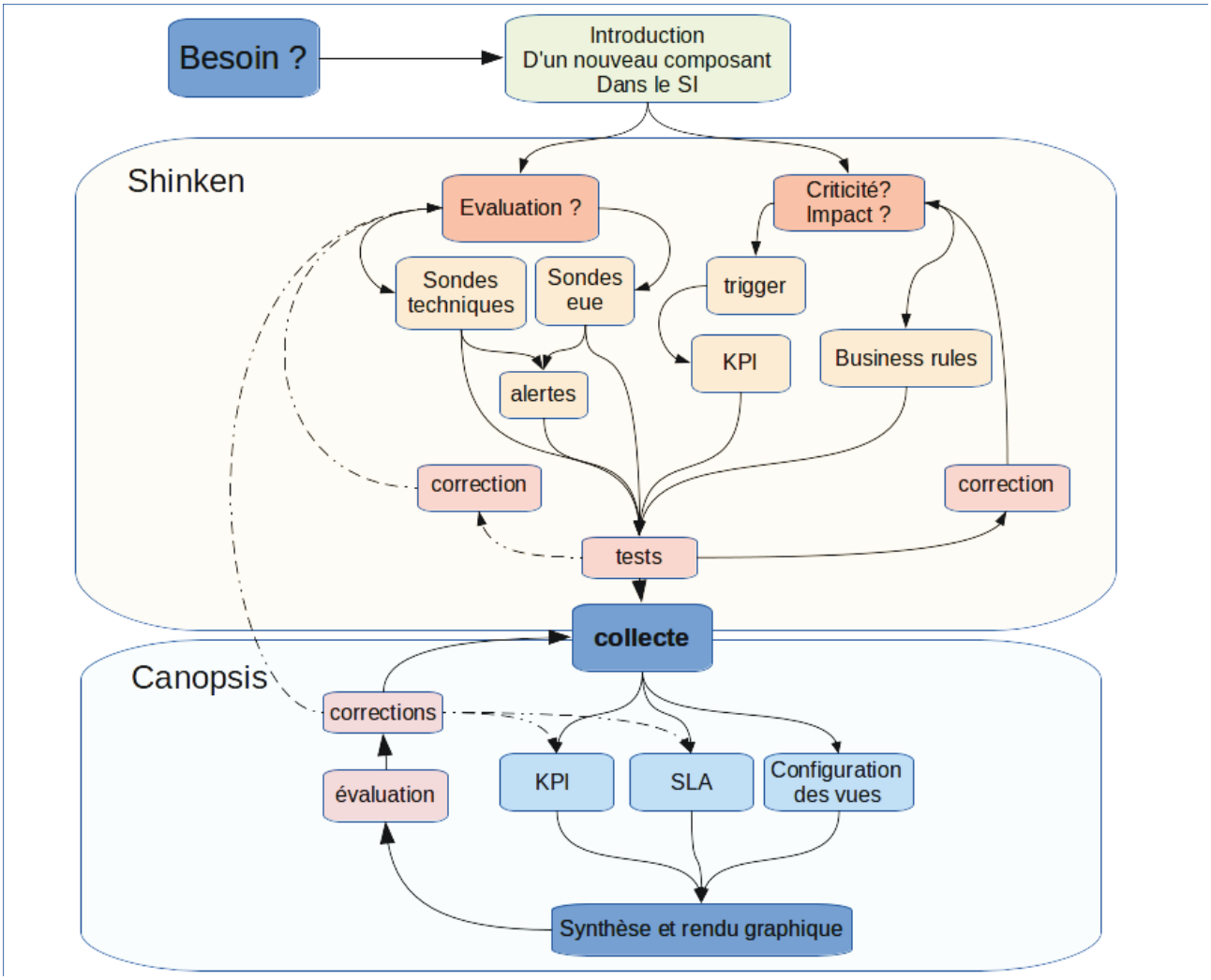


Illustration 6.1 : procédé itératif d'intégration d'éléments de supervision

6.7. L'IDE Sikuli

Illustration 6.2 : vue de l'IDE Sikuli

Illustration 6.3 : schéma des API Sikuli [128]

6.8. Synopsis SNMPTT

Illustration 6.4 : synopsis du daemon SNMPTT.

6.9. Exemples de vues Canopsis

Illustration 6.5 : rendu de scénarios de tests comportementaux

Illustration 6.6 : vue Canopsis pour l'application « imhoweb »

Illustration 6.7 : vue Canopsis de l'application Aravis

Illustration 6.8 : vue Canopsis - synthèse de l'état des applications

Illustration 6.9 : vue Canopsis - synthèse des liens réseau

7.Bibliographie

[1] I. Missaoui, « Cahier de recherche n° 5 : Valeur et performance des SI ». [En ligne]. Disponible sur: http://www.cigref.fr/cigref_publications/RapportsContainer/Parus2009/Valeur_et_performance_des_SI_CIGREF_2009.pdf.

[2] P. Bourdieu, *Esquisse d'une théorie de la pratique: précédé de trois études d'ethnologie kabyle*. Paris: Seuil, 2000.

[3] R. de Courcy, « Les systèmes d'information en réadaptation », *Réseau Int. CIDIH Facteurs Environnementaux*, vol. 1-2, n° 5, p. 7-10, 1992.

[4] P. Rabardel, *Les hommes et les technologies. Une approche cognitive des instruments contemporains*, Armand-Collin. Paris, 1995.

[5] D. Vinck, *Sociologie des sciences*. Paris: A. Colin, 1995.

[6] M. Crozier et E. Friedberg, *L'acteur et le système: les contraintes de l'action collective*. Paris: Editions du Seuil, 1977.

[7] D. Autissier et V. Delaye, *Mesurer la performance du système d'information*. Paris: Eyrolles, 2008.

[8] C. Gauzente, « MESURE DE LA PERFORMANCE GLOBALE DES Mesurer la performance des entreprises en l'absence d'indicateurs objectifs : quelle validité ? Analyse de la pertinence de certains indicateurs ». [En ligne]. Disponible sur: http://leg.u-bourgogne.fr/rev/032165.PDF.

[9] « Système — Wikipédia ». [En ligne]. Disponible sur: http://fr.wikipedia.org/wiki/Syst%C3%A8me. [Consulté le: 09-mars-2014].

[10] « Application performance management - Wikipedia, the free encyclopedia ». [En ligne]. Disponible sur: http://en.wikipedia.org/wiki/Application_performance_management. [Consulté le: 20-août-2014].

[11] « La demande de logement locatif social ». [En ligne]. Disponible sur: http://www.doubs.gouv.fr/Politiques-publiques/Amenagement-du-territoire-Construction-Logement-et-Transports/Politique-du-logement/La-gestion-sociale-du-logement/La-demande-de-logement-locatif-social. [Consulté le: 31-août-2014].

[12] « Single Euro Payments Area — Wikipédia ». [En ligne]. Disponible sur: http://fr.wikipedia.org/wiki/Single_Euro_Payments_Area. [Consulté le: 31-août-2014].

[13] P. Delbrayelle, « Fonctions ITIL et informatique en nuage », *ITIL France*. [En ligne]. Disponible sur: http://www.itilfrance.com/pages/docs/hgelun/pratique_fonctions_nuage.pdf. [Consulté le: 03-avr-2014].

[14] P. Delabrayelle, « ITIL France », *Le site francophone et gratuit sur l'ITSM*. [En ligne]. Disponible sur: http://www.itilfrance.com/. [Consulté le: 07-avr-2014].

[15] Great Britain et Office of Government Commerce, *ITIL, service design*. Norwich: TSO, the Stationery Office, 2011.

[16] « ITIL - Capacity_Management », *itil Library*. [En ligne]. Disponible sur: http://www.itlibrary.org/index.php?page=Capacity_Management. [Consulté le: 08-avr-2014].

[17] R. A. Steinberg, C. Rudd, S. Lacy, et A. Hanna, *ITIL service operation*. London: TSO, 2011.

[18] P. Delbrayelle, « ITIL V3 : Amélioration continue des services », *Itil France*. [En ligne]. Disponible sur: http://www.itilfrance.com/pages/docs/hgelun/itilv3_amelioration.pdf. [Consulté le: 13-avr-2014].

[19] « Urbanisation (informatique) — Wikipédia ». [En ligne]. Disponible sur: http://fr.wikipedia.org/wiki/Urbanisation_(informatique)#Exemple_de_r.C3.A8gles_de_d.C3.A9coupage_du_SI. [Consulté le: 14-avr-2014].

[20] « XenServer - Tech Info - Citrix ». [En ligne]. Disponible sur: http://www.citrix.fr/products/xenserver/tech-info.html. [Consulté le: 17-juin-2014].

[21] « Virtualisation des serveurs - Fonctionnalités de VMware vSphere | VMware France ». [En ligne]. Disponible

sur: http://www.vmware.com/fr/products/vsphere/features.html. [Consulté le: 27-déc-2013].
[22] « IBM Châssis BladeCenter H - France », 05-juin-2013. [En ligne]. Disponible sur: http://www-03.ibm.com/systems/fr/bladecenter/hardware/chassis/bladeh/. [Consulté le: 31-août-2014].
[23] C. Chaubal, « The architecture of VMWare ESXi ». [En ligne]. Disponible sur: http://www.vmware.com/files/pdf/ESXi_architecture.pdf. [Consulté le: 27-déc-2013].
[24] « Citrix XenServer - Logiciel de virtualisation de serveurs efficace - Citrix ». [En ligne]. Disponible sur: https://www.citrix.fr/products/xenserver/overview.html. [Consulté le: 04-janv-2014].
[25] « Zone démilitarisée (informatique) », *Wikipédia*. 02-avr-2014.
[26] « IOS (Cisco) — Wikipédia ». [En ligne]. Disponible sur: http://fr.wikipedia.org/wiki/IOS_(Cisco). [Consulté le: 17-juin-2014].
[27] « The NetBSD Project ». [En ligne]. Disponible sur: http://www.netbsd.org/. [Consulté le: 17-juin-2014].
[28] GIGREF, « L'Alignement stratégique du système d'information : Comment faire du système d'information un atout pour l'entreprise ? », 2002. [En ligne]. Disponible sur: http://cigref.typepad.fr/cigref_publications/RapportsContainer/Parus2002/2002_-_Alignement_strategique_du_systeme_d_information_web.pdf. [Consulté le: 20-avr-2014].
[29] G. Pujolle et O. Salvatori, *Les réseaux: édition 2003*. Paris: Eyrolles, 2002.
[30] « ASN.1 - Wikipédia ». [En ligne]. Disponible sur: http://fr.wikipedia.org/wiki/ASN.1. [Consulté le: 31-août-2013].
[31] « Introduction to ASN.1 ». [En ligne]. Disponible sur: http://www.itu.int/ITU-T/asn1/introduction/index.htm. [Consulté le: 31-août-2013].
[32] « Manpage of SNMPWALK ». [En ligne]. Disponible sur: http://www.net-snmp.org/docs/man/snmpwalk.html. [Consulté le: 05-mai-2014].
[33] « SNMP library for Python ». [En ligne]. Disponible sur: http://pysnmp.sourceforge.net/. [Consulté le: 05-mai-2014].
[34] « Net::SNMP - search.cpan.org ». [En ligne]. Disponible sur: http://search.cpan.org/~dtown/Net-SNMP-v6.0.1/lib/Net/SNMP.pm. [Consulté le: 05-mai-2014].
[35] P. Perez, « SNMP avancé », *L'élément commun CMISE, le service CMIS et le protocole CMIP*. [En ligne]. Disponible sur: http://www.docstoc.com/docs/110273701/D96-SNMP. [Consulté le: 26-avr-2014].
[36] « Web-Based Enterprise Management », *Wikipédia*. 19-août-2013.
[37] « WBEM (Web-Based Enterprise Management) ». .
[38] « CIM | DMTF ». [En ligne]. Disponible sur: http://dmtf.org/standards/cim. [Consulté le: 01-sept-2013].
[39] « Common Information Model », *Wikipédia*. 31-août-2013.
[40] « Vue d'ensemble du fournisseur WMI SNMP ». [En ligne]. Disponible sur: http://technet.microsoft.com/fr-fr/library/cc770487.aspx. [Consulté le: 01-sept-2013].
[41] « WMI and SQL (Windows) ». [En ligne]. Disponible sur: http://msdn.microsoft.com/en-us/library/aa394552(v=vs.85).aspx. [Consulté le: 06-mai-2014].
[42] « Check WMI Plus ». [En ligne]. Disponible sur: http://www.edcint.co.nz/checkwmiplus/. [Consulté le: 06-mai-2014].
[43] « Storage Management Initiative - Specification - Wikipédia ». [En ligne]. Disponible sur: http://fr.wikipedia.org/wiki/Storage_Management_Initiative_-_Specification. [Consulté le: 01-sept-2013].
[44] « ISO/IEC 24775:2011 - Information technology -- Storage management ». [En ligne]. Disponible sur: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=55234. [Consulté le: 07-mai-2014].
[45] « watir (Watir) · GitHub ». [En ligne]. Disponible sur: https://github.com/watir/. [Consulté le: 26-nov-2013].
[46] « Selenium - Web Browser Automation ». [En ligne]. Disponible sur: http://docs.seleniumhq.org/. [Consulté

le: 31-mai-2014].

[47] « WebInject - (HTTP) Web Application and Web Services Test Tool ». [En ligne]. Disponible sur: http://www.webinject.org/. [Consulté le: 31-mai-2014].

[48] « WWW::Mechanize - search.cpan.org ». [En ligne]. Disponible sur: http://search.cpan.org/~ether/WWW-Mechanize-1.73/lib/WWW/Mechanize.pm. [Consulté le: 31-mai-2014].

[49] « Sikuli Script - Home ». [En ligne]. Disponible sur: http://www.sikuli.org/. [Consulté le: 31-mai-2014].

[50] C. Tsung-Hsiang, T. Yeh, et R. C.Miller, « GUI Testing Using Computer Vision ». [En ligne]. Disponible sur: http://groups.csail.mit.edu/uid/projects/sikuli/sikuli-chi2010.pdf. [Consulté le: 20-nov-2013].

[51] « Monitoring and Management Using JMX Technology - Java SE Monitoring and Management Guide ». [En ligne]. Disponible sur: http://docs.oracle.com/javase/7/docs/technotes/guides/management/agent.html. [Consulté le: 20-août-2014].

[52] « yahoo/boomerang · GitHub ». [En ligne]. Disponible sur: https://github.com/yahoo/boomerang. [Consulté le: 20-août-2014].

[53] « Welcome to Jenkins CI! | Jenkins CI ». [En ligne]. Disponible sur: http://jenkins-ci.org/. [Consulté le: 06-sept-2014].

[54] « Application Performance Management - Supervision Applicative | Compuware APM ». [En ligne]. Disponible sur: http://www.compuware.com/fr_fr/application-performance-management.html. [Consulté le: 20-août-2014].

[55] « APM | Canopsis ». [En ligne]. Disponible sur: http://www.canopsis.com/Brick/apm. [Consulté le: 29-août-2014].

[56] « Prioritizing Gartner's APM Model | APMdigest ». [En ligne]. Disponible sur: http://apmdigest.com/prioritizing-gartners-apm-model. [Consulté le: 20-août-2014].

[57] « PhantomJS | PhantomJS ». [En ligne]. Disponible sur: http://phantomjs.org/. [Consulté le: 15-juin-2014].

[58] « Comparison of network monitoring systems », *Wikipedia, the free encyclopedia*. 14-avr-2014.

[59] « IT Infrastructure Monitoring, Operations Manager | HP® Official Site ». [En ligne]. Disponible sur: http://www8.hp.com/us/en/software-solutions/software.html?compURI=1170678#.U23gP3IRrCY. [Consulté le: 10-mai-2014].

[60] « IBM - Tivoli Monitoring ». [En ligne]. Disponible sur: http://www-03.ibm.com/software/products/en/tivomoni/. [Consulté le: 10-mai-2014].

[61] « Network Monitoring Software & Tools - CA Technologies ». .

[62] « IBM - Tivoli Netcool/OMNIbus - France ». [En ligne]. Disponible sur: http://www-03.ibm.com/software/products/fr/fr/ibmtivolinetcoolomnibus/. [Consulté le: 31-août-2013].

[63] « Homepage | POM MONITORING ». [En ligne]. Disponible sur: http://www.pom-monitoring.com/. [Consulté le: 08-mai-2014].

[64] « Homepage of Zabbix :: An Enterprise-Class Open Source Distributed Monitoring Solution ». [En ligne]. Disponible sur: http://www.zabbix.com/. [Consulté le: 10-mai-2014].

[65] « The OpenNMS Project ». [En ligne]. Disponible sur: http://www.opennms.org/. [Consulté le: 10-mai-2014].

[66] « Home Centreon ». [En ligne]. Disponible sur: http://www.centreon.com/. [Consulté le: 10-mai-2014].

[67] « Cacti® - The Complete RRDTool-based Graphing Solution ». [En ligne]. Disponible sur: http://www.cacti.net/. [Consulté le: 19-mai-2014].

[68] « Graphite - Scalable Realtime Graphing - Graphite ». [En ligne]. Disponible sur: http://graphite.wikidot.com/. [Consulté le: 19-mai-2014].

[69] « RRDtool - About RRDtool ». [En ligne]. Disponible sur: http://oss.oetiker.ch/rrdtool/. [Consulté le: 19-mai-2014].

[70] « Welcome to NagVis Home! - NagVis.org ». [En ligne]. Disponible sur: http://www.nagvis.org/. [Consulté le:

19-mai-2014].

[71] « Ganglia Monitoring System ». [En ligne]. Disponible sur: http://ganglia.sourceforge.net/. [Consulté le: 19-mai-2014].

[72] « Canopsis · capensis/canopsis Wiki · GitHub ». [En ligne]. Disponible sur: https://github.com/capensis/canopsis/wiki/Canopsis. [Consulté le: 10-mai-2014].

[73] « RealOpInsight - Open Source Business Service Monitoring Dashboard Toolkit for Nagios, Zabbix, Zenoss, Icinga, Shinken, op5, GroundWork, and Centreon ». [En ligne]. Disponible sur: http://realopinsight.com/. [Consulté le: 18-mai-2014].

[74] « Panorama [wiki monitoring-fr.org] ». [En ligne]. Disponible sur: http://wiki.monitoring-fr.org/supervision/links. [Consulté le: 19-mai-2014].

[75] « monitoringsucks/tool-repos · GitHub ». [En ligne]. Disponible sur: https://github.com/monitoringsucks/tool-repos. [Consulté le: 19-mai-2014].

[76] « Comparison of network monitoring systems - Wikipedia, the free encyclopedia ». [En ligne]. Disponible sur: http://en.wikipedia.org/wiki/Comparison_of_network_monitoring_systems. [Consulté le: 19-mai-2014].

[77] « Nagios - The Industry Standard in IT Infrastructure Monitoring », *Nagios*. [En ligne]. Disponible sur: http://www.nagios.org/. [Consulté le: 31-août-2013].

[78] « Nagios Exchange ». [En ligne]. Disponible sur: http://exchange.nagios.org/. [Consulté le: 31-août-2013].

[79] « op5 | Open Source Network Monitoring | op5 Server Monitoring ». [En ligne]. Disponible sur: http://www.op5.com/. [Consulté le: 11-mai-2014].

[80] « Interview de Jean Gabes, le créateur de Shinken | Le blog de NicoLargo ». [En ligne]. Disponible sur: http://blog.nicolargo.com/2011/08/interview-de-jean-gabes-le-createur-de-shinken.html. [Consulté le: 11-mai-2014].

[81] « Où va Nagios ? | Communauté Francophone de la Supervision Libre ». [En ligne]. Disponible sur: http://www.monitoring-fr.org/2010/06/ou-va-nagios/. [Consulté le: 11-mai-2014].

[82] J. Gabès, « Shinken : The next Industry Standard in IT Monitoring », *Shinken Monitoring*. [En ligne]. Disponible sur: http://www.shinken-monitoring.org/. [Consulté le: 31-août-2013].

[83] J. Gabès, « naparuba/shinken · GitHub ». [En ligne]. Disponible sur: https://github.com/naparuba/shinken. [Consulté le: 11-mai-2014].

[84] « Nagios - Browse /nrpe-2.x at SourceForge.net ». [En ligne]. Disponible sur: http://sourceforge.net/projects/nagios/files/nrpe-2.x/. [Consulté le: 11-mai-2014].

[85] « NSCA - Nagios Service Check Acceptor - Nagios Exchange ». [En ligne]. Disponible sur: http://exchange.nagios.org/directory/Addons/Passive-Checks/NSCA--2D-Nagios-Service-Check-Acceptor/details. [Consulté le: 11-mai-2014].

[86] M. Schubert, Éd., *Nagios 3 enterprise network monitoring: including plug-ins and hardware devices*. Burlington, MA: Syngress Pub, 2008.

[87] « Chapitre 62. Régler Nagios pour des performances maximales ». [En ligne]. Disponible sur: http://doc.monitoring-fr.org/3_0/html/securityandperformancetuning-tuning.html. [Consulté le: 14-mai-2014].

[88] « Shinken : quand un Python rencontre Nagios / GLMFHS-049 / GNU/Linux Magazine / Connect - Edition Diamond ». [En ligne]. Disponible sur: http://connect.ed-diamond.com/GNU-Linux-Magazine/GLMFHS-049/Shinken-quand-un-Python-rencontre-Nagios. [Consulté le: 14-mai-2014].

[89] « MK Livestatus ». [En ligne]. Disponible sur: http://mathias-kettner.de/checkmk_livestatus.html. [Consulté le: 18-mai-2014].

[90] « Thruk Monitoring Webinterface ». [En ligne]. Disponible sur: http://www.thruk.org/. [Consulté le: 18-mai-2014].

[91] « Splunk | Gestion des applications, Opérations informatiques de gestion, Sécurité et conformité ». [En ligne].

Disponible sur: http://fr.splunk.com/. [Consulté le: 18-mai-2014].

[92] « livestatus_shinken [Shinken] ». [En ligne]. Disponible sur: http://www.shinken-monitoring.org/wiki/livestatus_shinken. [Consulté le: 18-mai-2014].

[93] « Architecture dirigée par les données — Wikipédia ». [En ligne]. Disponible sur: http://fr.wikipedia.org/wiki/Architecture_dirig%C3%A9e_par_les_donn%C3%A9es. [Consulté le: 18-mai-2014].

[94] J. Gabès, « Python, le serpent très dynamique.GNU-Linux Magazine HS n°49 », p. 14-20, 2010.

[95] « shinken-monitoring/mod-webui · GitHub », *dépôt Github*. [En ligne]. Disponible sur: https://github.com/shinken-monitoring/mod-webui. [Consulté le: 30-mai-2014].

[96] « shinken-monitoring/mod-ws-arbiter · GitHub ». [En ligne]. Disponible sur: https://github.com/shinken-monitoring/mod-ws-arbiter. [Consulté le: 30-mai-2014].

[97] « shinken-monitoring/mod-mongodb · GitHub ». [En ligne]. Disponible sur: https://github.com/shinken-monitoring/mod-mongodb. [Consulté le: 30-mai-2014].

[98] « shinken-monitoring/mod-canopsis · GitHub ». [En ligne]. Disponible sur: https://github.com/shinken-monitoring/mod-canopsis. [Consulté le: 30-mai-2014].

[99] « Graphite - Scalable Realtime Graphing - Graphite ». [En ligne]. Disponible sur: http://graphite.wikidot.com/. [Consulté le: 31-août-2013].

[100]« kombu 3.0.16 : Python Package Index », *Messaging library for Python*. [En ligne]. Disponible sur: https://pypi.python.org/pypi/kombu. [Consulté le: 31-mai-2014].

[101]« Home | AMQP ». [En ligne]. Disponible sur: http://amqp.org/. [Consulté le: 31-mai-2014].

[102]« Welcome to NagVis Home! - NagVis.org ». [En ligne]. Disponible sur: http://www.nagvis.org/. [Consulté le: 19-mai-2014].

[103]« Operational Data Store — Wikipédia ». [En ligne]. Disponible sur: http://fr.wikipedia.org/wiki/Operational_Data_Store. [Consulté le: 06-sept-2014].

[104]« Redis ». [En ligne]. Disponible sur: http://redis.io/. [Consulté le: 12-juin-2014].

[105]« Homepage | Celery: Distributed Task Queue ». [En ligne]. Disponible sur: http://www.celeryproject.org/. [Consulté le: 11-juin-2014].

[106]« Réinvention de la gestion des informations | MongoDB ». [En ligne]. Disponible sur: http://www.mongodb.com/fr. [Consulté le: 09-juin-2014].

[107]« Just a little Python: GridFS: The MongoDB Filesystem ». [En ligne]. Disponible sur: http://blog.pythonisito.com/2012/05/gridfs-mongodb-filesystem.html. [Consulté le: 31-oct-2013].

[108]« Gunicorn - Python WSGI HTTP Server for UNIX ». [En ligne]. Disponible sur: http://gunicorn.org/. [Consulté le: 12-juin-2014].

[109]« PEP 3333 -- Python Web Server Gateway Interface v1.0.1 ». [En ligne]. Disponible sur: http://legacy.python.org/dev/peps/pep-3333/. [Consulté le: 12-juin-2014].

[110]R. T. Fielding, « Architectural Styles and the Design of Network-based Software Architectures :CHAPTER 5: Representational State Transfer (REST) ». [En ligne]. Disponible sur: http://www.ics.uci.edu/%7Efielding/pubs/dissertation/rest_arch_style.htm. [Consulté le: 15-juin-2014].

[111]« JavaScript Framework for Building Rich Desktop Web Applications | Sencha Ext JS | Products | Sencha ». [En ligne]. Disponible sur: http://www.sencha.com/products/extjs/. [Consulté le: 09-juin-2014].

[112]« jQuery ». [En ligne]. Disponible sur: http://jquery.com/. [Consulté le: 09-juin-2014].

[113]« Flot: Attractive JavaScript plotting for jQuery ». [En ligne]. Disponible sur: http://www.flotcharts.org/. [Consulté le: 09-juin-2014].

[114]« SNMP library for Python ». [En ligne]. Disponible sur: http://pysnmp.sourceforge.net/. [Consulté le: 12-sept-2013].

[115]« Whisper FAQ - Graphite ». [En ligne]. Disponible sur: http://graphite.wikidot.com/whisper. [Consulté le: 22-juin-2014].

[116]« The URL API — Graphite 0.9.9 documentation ». [En ligne]. Disponible sur: http://graphite.readthedocs.org/en/1.0/url-api.html. [Consulté le: 22-juin-2014].

[117]« CasperJS, a navigation scripting and testing utility for PhantomJS and SlimerJS ». [En ligne]. Disponible sur: http://casperjs.org/. [Consulté le: 22-juin-2014].

[118]« XML-RPC », *Wikipédia*. 28-juill-2014.

[119]X. Dusart, « SNMP et Nagios ». [En ligne]. Disponible sur: http://xavier.dusart.free.fr/nagios/snmp-traps.html. [Consulté le: 15-juin-2014].

[120]A. Burger, « SNMPTT », *SNMP Trap Translator*. [En ligne]. Disponible sur: http://www.snmptt.org/about.shtml. [Consulté le: 31-août-2013].

[121]« RFC 5424 - The Syslog Protocol ». [En ligne]. Disponible sur: http://tools.ietf.org/html/rfc5424. [Consulté le: 22-juin-2014].

[122]« Cucumber - Making BDD fun ». [En ligne]. Disponible sur: http://cukes.info/. [Consulté le: 31-mai-2014].

[123]« Traitement automatique du langage naturel », *Wikipédia*. 26-août-2014.

[124]« Behavior Driven Development — Wikipédia ». [En ligne]. Disponible sur: http://fr.wikipedia.org/wiki/Behavior_Driven_Development. [Consulté le: 31-mai-2014].

[125]L. Lévy-Bencheton, *L'infotechnocratie: le déni de complexité dans l'informatique*. Paris: Hermès science publications-Lavoisier, 2012.

[126]P. par P. L. le 10 décembre 2013, « La conduite du changement », *Alcyonix*. [En ligne]. Disponible sur: http://www.alcyonix.com/tribunes/la-conduite-du-changement/. [Consulté le: 29-août-2014].

[127]« Nagios — Wikipédia ». [En ligne]. Disponible sur: http://fr.wikipedia.org/wiki/Nagios. [Consulté le: 11-mai-2014].

[128]sikuli doc team, « How Sikuli Works — Sikuli X 1.0 documentation ». [En ligne]. Disponible sur: http://doc.sikuli.org/devs/system-design.html. [Consulté le: 26-août-2014].

8.Index des illustrations

Résumé

En matière de gestion des systèmes d'information, la supervision est un sujet très couru, ainsi qu'en témoignent ses nombreux soutènements, qu'il s'agisse de préconisations, de normes, de modèles ou d'implémentations. De fait, superviser suppose d'appréhender d'un système d'information dans sa globalité, c'est-à-dire comme une composante organisationnelle à part entière. Entre l'usage et la technicité, le système d'information concentre une multiplicité d'approches et de points de vue qui expliquent sa complexité intrinsèque. Nous observons ici comment, et à quelles conditions, les outils de supervision procurent un moyen de restituer à ce système complexe une cohérence globale incluant ses aspects techniques et fonctionnels.

Mots-clés : système d'information, supervision, hypervision, sondes techniques, sondes fonctionnelles, indicateurs-clés de performance.

Abstract

When it comes to information systems management, supervision is a popular topic, as indicated by the numerous existing foundations such as recommendations, models, standards and implementations. Indeed, supervising implies understanding an information system in its totality, that is to say, as an organisational component in itself. Between technicity and usage, the information system comprises a multitude of approaches and points of view which explain its intrinsic complexity. This paper explores how, and under which conditions, supervision tools provide a way of rendering an overall coherence to this complex system, including its technical and functional aspects.

Keywords : information system, supervision, hypervision, technical checks, end user experience, key performance indicators.