MASTER EN SCIENCES INFORMATIQUES

Développement d'un tableau de bord de gestion d'une salle machine

Vander Elst, Etienne

Award date:
2014

Awarding institution:
Universite de Namur

Université de Namur
Faculté d'informatique
Année académique 2013–2014

Développement d'un tableau de bord de gestion d'une salle machine

Vander Elst Etienne

Promoteur : _____ (Signature pour approbation du dépôt - REE art. 40)
Jean-Noël Colin

Mémoire présenté en vue de l'obtention du grade de
Master en Sciences Informatiques.

Résumé

Représentation d'une infrastructure informatique suivant une approche par graphe de dépendances pour un tableau de bord de gestion de salle machine. Nous exposons les besoins d'une telle solution et les bénéfices pour les administrateurs système mais aussi pour les managers. Nous générons un graphe de dépendances sur base d'une structure en calques utile et intuitive. Nous exploitons ce graphe pour effectuer des analyses de causes racines et d'impacts. Nous proposons l'architecture d'une solution qui communique avec un système de surveillance informatique existant et plusieurs techniques d'identification automatique des dépendances dans un environnement distribué.

Mots-clé : tableau de bord, graphe de dépendances, services, administration système, gestion

Abstract

A dependency graph-based approach to represent an IT infrastructure in a system administration dashboard. We expose the needs for such a solution and the benefits for system administrators as well as for IT managers. We generate a dependency graph based on a useful and intuitive layered structure and use this graph to perform Root Causes Analysis and Impact Analysis. We suggest the architecture of a solution interfacing with an existing monitoring system and several techniques to automatically identify dependencies in a distributed environment.

Keywords : dashboard, dependency graph, services, system administration, management

Avant-propos

Sans cesse en recherche de nouveaux défis, je suis fasciné par les sciences informatiques. Leur évolution exponentielle, l'énorme diversité de leurs applications et la masse de connaissance qu'elles nous permettent d'apprendre et comprendre sont, à mon sens, des qualités que peu d'autres sciences peuvent égaler.

Élaborer une solution qui permette d'administrer, d'organiser et de huiler les rouages qui mettent en œuvre les activités des sciences informatiques est, pour moi, un objectif dont la quête commence avec ces quelques premiers pas.

Je souhaite tout d'abord remercier mon promoteur, le professeur Jean-Noël Colin de m'avoir permis de travailler sur ce sujet fascinant et d'apporter ma contribution aux sciences informatiques. Je le remercie pour son suivi et ses conseils d'orientation tout au long de ce travail.

Je remercie particulièrement Jean-Marc André d'avoir partagé avec moi son expertise et son expérience dans le domaine. Je le remercie pour ses conseils ainsi que la mise à disposition d'une infrastructure pour les besoins de la solution.

Je remercie les membres du jury de prendre le temps d'évaluer mon travail.

Je remercie le corps professoral de l'Université de Namur pour les excellents cours qui m'ont été dispensés.

Je remercie mes collègues de classe pour l'agréable ambiance qui a régné durant ces années d'études.

Enfin, je remercie mes proches pour leur compréhension, leur soutient et la relecture de ce mémoire.

«Pour citer Whitman : « [...] Qu'y a-t-il de bon en cela ? Ô moi ! Ô la vie ! ».
Réponse : que tu es ici, que la vie existe, et l'identité.
Que le prodigieux spectacle continue et que tu peux y apporter ta rime...
Quelle sera votre rime ?»

Robin Williams, *Le Cercle des poètes disparus* (1989),
écrit par Tom Schulman

Table des figures

Table des matières

Introduction

A l'heure où les systèmes d'information sont au cœur de l'activité de nombreuses disciplines, il est important de pouvoir, à tout moment, être informé de l'état de santé de ces systèmes. La disponibilité et les performances des systèmes d'information sont vitales dans de nombreux domaines tels que la médecine, la finance, la défense nationale, l'aéronautique et spatiale, etc. Être immédiatement alerté d'une défaillance pour réduire le délai de restauration du service, anticiper une panne à partir de ses symptômes, ou encore mettre en évidence les défauts d'architecture, voilà ce que nous apportent les tableaux de bord de gestion de salle machine, aussi appelés "dashboard" en anglais.

Dans les chapitres qui suivent, nous verrons qu'il est possible d'étendre l'utilité de ces tableaux de bord en leur permettant de répondre à la question d'impact d'une défaillance sur l'intégralité du système.

La fonction d'administrateur systèmes est critique pour assurer le bon fonctionnement des systèmes d'information. L'évolution des technologies matérielles et logicielles est indéniable et celles-ci répondent aisément aux besoins des activités de notre société. Cependant, l'implication de l'informatique dans la société est telle que les administrateurs ne peuvent plus se limiter à l'aspect technique des systèmes. Il leur est nécessaire de savoir et comprendre en quoi l'existence de tel ou tel composant a un sens pour l'activité de l'entreprise.

Les solutions de tableaux de bord sont nombreuses et très efficaces quand il s'agit de présenter l'aspect technique et les dépendances implicites entre les systèmes. Au travers de ce travail de recherche et de conception, nous avons souhaité leur apporter la connaissance des interactions et des dépendances qui dépasseront la notion technique par la dimension sémantique.

Si nous prenons l'exemple d'un quelconque appareil électrique, son bon fonctionnement dépend essentiellement de ses propres composants et de son alimentation. Cette dépendance est forte et évidente : l'absence d'alimentation électrique aura pour seule et unique conséquence l'arrêt complet de l'appareil. Dans ce cas de figure, peu importe que ce soit l'alimentation électrique ou seul l'appareil qui soit surveillé, la panne sera détectée.

Prenons maintenant l'exemple d'un serveur de messagerie et de sa dépendance à une connexion internet. La défaillance de la connexion n'aura cette fois plus pour conséquence l'arrêt du serveur. Celui-ci ne pourra plus remplir pleinement sa fonction, mais ne sera pas considéré hors-service pour autant. S'il est facile de détecter un dysfonctionnement lors de l'envoi d'un message, il en est beaucoup moins pour la réception. En effet, l'absence de messages entrants peut avoir bien d'autres significations. Il est donc, dans ce cas, indispensable, non seulement de surveiller la connexion et le serveur, mais également de connaitre le lien de dépendance qui les relie.

Bien entendu, ce type de dépendance est bien connue des administrateurs systèmes puisqu'elle entre pleinement dans le domaine des compétences informatiques, mais peut-on en dire autant de celle qui lie un logiciel de comptabilité au système qui imprime et expédie les factures aux clients ?

Si une panne peut sembler anodine au premier abord, les conséquences qu'elle aura sur l'ensemble du système peuvent être critiques pour l'entreprise. L'arbitrage des priorités lorsque plusieurs défaillances surviennent simultanément peut, dès lors, requérir une connaissance avancée de l'infrastructure, des activités de la société et, souvent, de l'avis de plusieurs intervenants.

Plutôt que de demander d'un administrateur qu'il soit omniscient et dispose d'une mémoire parfaite, il est plus réaliste de concevoir un système, un tableau de bord, qui soit conscient des dépendances qui lient chaque élément du système informatique.

Dans cet exercice de conception et développement d'un tableau de bord, nous avons souhaité répondre à la problématique de normalisation de représentation des éléments d'une salle machine et leurs dépendances dans l'optique d'apporter une réponse rapide et efficace à la question d'impact d'une défaillance sur l'ensemble du système.

Les relations entre les éléments sont le point central de la problématique puisque ce sont elles qui impliquent et déterminent des dépendances. Nous exploiterons la théorie des graphes qui nous permet de modéliser efficacement cette problématique et nous apporte les outils adéquats à l'exécution d'analyse de causes racines et d'impact.

Pour ce faire, il conviendra de définir une méthode de modélisation universelle ainsi que les algorithmes et techniques qui analyseront cette représentation. Nous verrons également que nous pouvons exploiter cette modélisation à d'autres fins que l'aide à la décision en cas de panne.

Nous commencerons par définir les critères d'efficacité d'un tableau de bord, nous regarderons aux solutions qui existent, et ferons un rappel sur la théorie des graphes. Nous nous intéresserons aux recherches qui ont été faites dans le domaine de l'administration système et exploitant des graphes de dépendances.

Nous poursuivrons par l'élaboration d'une représentation adéquate d'une salle machine, orientée pour répondre aux besoins énoncés. Et nous détaillerons les différentes utilités et les méthodes d'exploitation de cette représentation.

Enfin, nous proposerons un exemple d'implémentation fonctionnel et observerons son intégration dans un environnement distribué existant.

Première partie

Etat de l'art

Chapitre 1

Tableaux de bord et systèmes de surveillance

1.1 Qu'est-ce qu'un tableau de bord

Un tableau de bord de gestion d'une salle machine est une représentation de celle-ci qui exprime son état de santé et d'activité. C'est un outil d'aide à la décision mais également de pilotage. C'est la façade du système de surveillance de l'infrastructure. Si sa fonction première est d'informer, il peut également fournir certaines fonctionnalités d'interaction avec les éléments de la salle machine telles qu'exécuter une commande ou démarrer une session d'accès distant. [QualNet, 2013]

C'est également un moyen d'être alerté lorsqu'un composant de l'infrastructure est défaillant ou que des erreurs surviennent.

La fonction d'administrateur système se constitue essentiellement de deux prérogatives :

1. Assurer un fonctionnement optimal de l'infrastructure informatique conformément aux besoins de l'entreprise
2. Assurer des pertes minimales en cas de dysfonctionnement

Un tableau de bord est donc un outil essentiel pour la première mais également pour la seconde puisqu'il permet de réduire le délai de détection et d'analyse, et même d'agir anticipativement sur une panne qui s'annonce avant qu'elle ne survienne.

Si certains tableaux de bord pourraient se contenter d'être une représentation de l'infrastructure, c'est quand ils sont couplés à un système de surveillance qu'ils prennent toute leur utilité. De même que l'utilité d'une base d'information d'état des systèmes est réduite sans une sur-couche de visualisation. C'est pourquoi les solutions actuelles combinent généralement tableaux de bord et système de surveillance.

Il va sans dire que, ces tableaux de bords exposant bon nombre d'informations sur l'infrastructure informatique, la sécurité d'accès est de rigueur.

Il conviendra de définir les limites qui définissent le système, tant en termes de couverture que de détails de représentation. La section 7.1 s'applique à répondre à cette problématique.

1.2 Critères d'efficacité d'un tableau de bord

Outre la localisation d'un dysfonctionnement, son impact variera suivant
- le délai de détection et d'alerte
- le délai d'analyse et de décision
- le délai d'intervention

Au travers des tableaux de bord, nous pouvons agir sur les deux premiers délais.

Observation continue [Wikipedia, 2014e] [Wikipedia, 2014d]

La surveillance informatique ou "system monitoring" (en anglais), est une discipline qui est rapidement apparue avec la naissance de l'informatique. Il était, en effet, essentiel de s'assurer que le matériel ou logiciel réalisé effectuait bien ce qui était prévu et comme cela était prévu. Les constructeurs et développeurs ont donc mis en place des interfaces d'observation sur les systèmes conçus.

Suivant les systèmes, nous retrouvons différentes interfaces :
- **Log/Journal** : au fur et à mesure de son fonctionnement, le système écrit dans un fichier toute information susceptible de faire l'objet d'une vérification. On y retrouve généralement des informations de début et de fin de traitement, des avertissements de potentiels problèmes, mais aussi les erreurs rencontrées.
- **Management Information Base (MIB)** : une MIB est une base d'information structurée en arborescence hiérarchique, décrite dans les RFC1155 [1] et RFC1213 [2]. Elle contient de nombreuses informations détaillées sur le système, ses composants, leur configuration et leur statut. Elle est spécialisée dans le modèle TCP/IP et est présente dans tous les équipements réseaux actuels, incluant serveurs, routeurs, imprimantes réseaux, etc. [Wikipedia, 2014c]
- **Network Management Interface Card (NMIC)** : les NMIC sont des cartes réseaux dédiées à la surveillance et l'écoute du réseau. Elles sont utilisées pour minimiser l'utilisation de ressources réseaux par le système de surveillance sur le réseau de production. [Wikipedia, 2013]
- **Port mirroring** : le port mirroring est une technique employée sur les équipements réseaux (i.e. routeurs, switchs) et consiste à répliquer le trafic qui traverse un port sur un second port. Celui-ci est directement exploité par le système de surveillance pour analyser le trafic.
- **Solutions propriétaires** : certains constructeurs incluent des solutions propres et optimisées pour leurs produits (e.g. Dell OpenManage [3], DRAC [4]).

Nous pouvons accéder à ces interfaces au travers de plusieurs protocoles dont certains sont essentiellement dédiés à la surveillance :

1. RFC1155 Structure and Identification of Management Information for TCP/IP-based Internets, `http://tools.ietf.org/html/rfc1155`
2. RFC1213 Management Information Base for Network Management of TCP/IP-based internets : MIB-II, `http://tools.ietf.org/html/rfc1213`
3. `http://en.community.dell.com/techcenter/systems-management/w/wiki/1757.dell-openmanage-systems-management-tools.aspx`
4. Dell Remote Access Controller `http://en.community.dell.com/techcenter/systems-management/w/wiki/3204.dell-remote-access-controller-drac-idrac.aspx`

- **Simple Network Monitoring Protocol (SNMP)** : décrit dans le RFC1157[5], le protocole SNMP est prévu pour interroger une MIB. C'est un protocol de la couche 7 du modèle OSI[6], la couche Application. Un client SNMP peut envoyer une requête à un agent SNMP afin d'obtenir l'information se trouvant à un nœud précis de l'arborescence, identifié par un Object Identifier (OID)[7].
- **Common Management Information Protocol (CMIP)** également de la couche 7 du modèle OSI et en concurrence directe avec SNMP, le protocole CMIP est défini dans la recommandation ITU-T X.711[8] et ISO/IEC9596-1[9]. A l'instar d'autres protocoles, il requiert la présence d'un agent sur le système surveillé. Il est plus complet que SNMP en permettant, par exemple, plus de flexibilité dans la modification du système surveillé. Il est cependant bien moins présent dans les équipement TCP/IP car il est plus complexe et consomme plus de ressources. [Wikipedia, 2014a]
- Les applicatifs peuvent disposer de protocoles de surveillance et de gestion propre (e.g. CORBA[10], JMX[11], etc.).
- Notons que la plupart des protocoles de communication disposent d'outils qui peuvent être exploités par les systèmes de surveillance (i.e. ping, traceroute pour TCP/IP, response code pour HTTP, etc.).

Dans la surveillance informatique, quatre méthodes sont utilisées pour récupérer de l'information sur un système. Cette information est ensuite analysée pour déterminer l'état de santé du système. Suivant le résultat de l'analyse, plus d'actions peuvent être entreprises par le système de surveillance pour confirmer et/ou analyser plus avant la défaillance.

Dans la première (figure 1.1 (1)), le système à surveiller alimente une base de donnée ou un fichier maintenu par un serveur tiers. Le système de surveillance le consulte ensuite pour récupérer l'information désirée. Plusieurs applicatifs existent à cette fin, citons par exemple syslog, rabbitmq ou encore scribe.

Dans la seconde (figure 1.1 (2)), le système de surveillance interroge directement l'élément surveillé. C'est le cas de SNMP, CMIP, etc.

Dans la troisième (figure 1.1 (3)), c'est le système surveillé qui envoie une notification soit à intervalle régulier (e.g. heartbeat[12]) soit lorsqu'un évènement déterminé se produit. Le message peut également contenir des informations sur l'état de santé, directement exploitées par le système de surveillance.

La quatrième méthode consiste à régulièrement fournir des données d'entrée à une application et vérifier que les données de sortie correspondent à ce qui est attendu. Nous pouvons, par exemple, faire transiter un message entre deux boites mail et vérifier que le destinataire ré-

5. RFC1157 A Simple Network Management Protocol (SNMP), `http://tools.ietf.org/html/rfc1157`
6. Open Systems Interconnection model
7. OID `https://standards.ieee.org/develop/regauth/tut/oid.pdf`
8. ITU-T Recommandation X.711 `http://www.itu.int/rec/T-REC-X.711/en/`
9. ISO/IEC9596-1 `http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=29698`
10. Common Object Request Broker Architecture,`http://www.omg.org/spec/CORBA/Current/`
11. Java Management Extensions, `http://www.oracle.com/technetwork/java/javase/tech/javamanagement-140525.html`
12. Technique selon laquelle un message préci est envoyé à un intervalle de temps régulier. L'absence de ce message durant un délai imparti déclare l'émetteur comme étant défectueux.

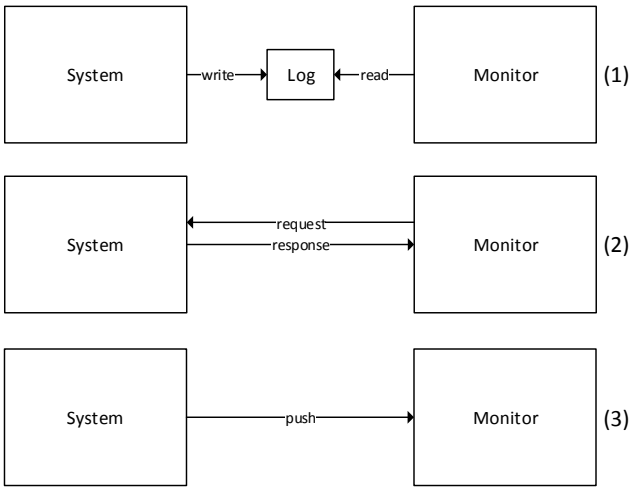

FIGURE 1.1 – Méthodes de surveillance

ceptionne correctement le message dans un délai acceptable. Nous pouvons également observer les messages qui transitent dans un réseau afin de vérifier que leur structure est correcte.

Tous les éléments d'une salle machine ne peuvent pas être surveillés de la même façon. Tous n'offrent pas les mêmes interfaces et fonctionnalités. Le système de surveillance sera donc configuré de telle manière qu'il appliquera la méthode adéquate à chaque élément pour récupérer les informations nécessaires à établir un diagnostic. Une fois l'état de santé établi et suivant le contexte et sa configuration, il saura quelle action entreprendre et, le cas échéant, qui alerter et par quel moyen.

La détection d'une panne doit s'opérer lorsqu'elle se produit et non pas lorsque le service concerné est utilisé. La meilleure façon de détecter une panne dans un délai restreint est l'observation continue. C'est également un bon moyen d'avoir une mesure des performances des équipements et applications sur une période de temps.

Gestion d'évènements

Une fois les données d'observation rassemblées par le système de surveillance, le tableau de bord peut les exploiter, les analyser et les présenter.

Tout changement d'état, quel qu'il soit, est résumé à la notion d'évènement. C'est par sa configuration que le tableau de bord catégorisera ces évènements soit en panne, soit en symptôme, soit en simple information.

La gestion des évènements peut générer différents types d'alerte [Dragich, 2012] :

- Réactive : une alerte réactive signifie que l'évènement nécessite une réaction immédiate : une dégradation ou une interruption de service s'est produite. C'est le cas lorsque l'évènement est une panne qui a une incidence immédiate sur la production.
- Proactive : dans le cas où l'évènement est une panne mais n'a pas d'incidence directe sur la production, l'alerte est proactive. L'apparition d'un second évènement peut dé-

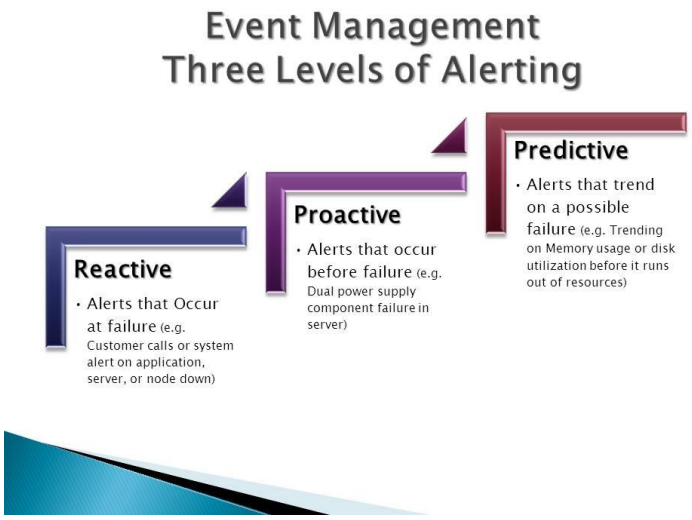

FIGURE 1.2 – Event Management - Three levels of alerting [Dragich, 2012]

couler sur une interruption de disponibilité. Par exemple, lorsqu'une des alimentations d'un serveur disposant d'une double alimentation tombe en panne, celui-ci continue de fonctionner normalement. L'existence d'une panne est confirmée et une intervention est nécessaire avant que la seconde alimentation ne s'arrête, mais pour le moment, aucune interruption de service ne s'est produite.

– Prédictive : une alerte prédictive est générée lorsque l'évènement n'est pas une panne mais un changement d'état qui dénonce une possible future panne. Typiquement, lorsque l'utilisation d'une ressource atteint un seuil proche des 100%.

Analyse de panne et aide à la décision [USNRC, 1998, Ch. I-1]

Quel que soit le système, en cas de panne dont les effets persistent, la décision finale doit revenir à l'être humain. Si aucune action corrective satisfaisante ne peut être entreprise par l'infrastructure elle-même (e.g. redémarrage de service, basculement de ressources ou de liens, etc.), le dysfonctionnement requerra une analyse de la part d'un humain. Les infrastructures peuvent aider la prise de décision par l'apport d'informations contextuelles.

Pour être complète, une information doit être contextualisée. Ainsi, lorsqu'il est question de générer une alerte, celle-ci doit être accompagnée de toute donnée relative à l'évènement qui se produit ainsi que toutes celles susceptibles d'être requises lorsqu'une décision doit être prise ; notamment, l'état de fonctionnement des éléments proches de celui concerné par l'alerte. [Microsoft, 2014]

Nous ne pouvons pas imaginer un monde où aucune décision n'est prise tant que toutes les informations ne sont pas disponibles. L'urgence des situations généralement rencontrées nous impose des décisions rapides. Cette dernière est directement proportionnelle à la criticité et à l'ampleur des conséquences possibles de la panne. Cette contrainte temporelle est ce qui distingue une bonne décision d'une décision correcte. L'expérience de l'administrateur système joue souvent un rôle important dans l'analyse du problème et la prise de décision. Mais tout le monde ne dispose pas d'une telle expérience.

Pour prendre une décision correcte, il nous faut :
- L'identification des informations qui seraient pertinentes pour la décision
- Un système dédié à l'acquisition de ces informations (i.e. le système de surveillance)
- Une analyse rationnelle des données

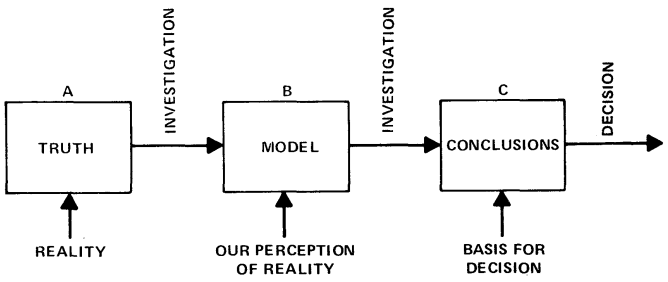

FIGURE 1.3 – Relationship between Reality, System Model, and Decision Process [USNRC, 1998, Ch. I-1]

La figure 1.3 décrit la relation entre la réalité, l'interprétation de cette réalité et la prise de décision.

Dans notre cas, la première investigation produisant le modèle est réalisée par le système de surveillance. La seconde est l'œuvre du tableau de bord. Nous voyons qu'il est essentiel que le modèle produit par la surveillance soit le plus fidèle possible à la réalité. Il nous faut mettre l'accent sur l'identification des informations pertinentes et l'acquisition de celles-ci.

Le tableau de bord doit analyser, dans un laps de temps limité, l'évènement – catégorisé en temps que panne – qu'il vient de détecter. Il existe plusieurs méthodes d'analyse qui seront décrites dans le chapitre 3. Il doit présenter toutes les informations pertinentes de sorte que la prise de décision soit rapide, correcte et ne nécessite pas ou peu d'analyses supplémentaires. Dans le chapitre 6, nous appliquerons une analyse. La partie II ("Conception de la solution") détaillera la conception et proposera une représentation qui répond à ces critères.

Afin d'accélérer la prise de décision, il serait également intéressant de pouvoir référencer une base de connaissance correspondant au problème, des documents internes (e.g. EBIOS[13], plan d'action, etc.), ou encore des informations de contact.

Cohérence et simplicité [Ramdoyal, 2012]

S'il est important de présenter toutes les informations, il est essentiel que la présentation soit simple, claire et cohérente. La lecture doit être aisée et précise. Pour réduire le délai d'analyse et de décision, l'administrateur doit pouvoir saisir l'incident et l'ampleur de la situation en quelques coups d'œil. Un écran surchargé d'informations porte à confusion et demande inutilement plus de concentration.

Les tableaux de bords sont principalement des interfaces visuelles. Il faut donc proscrire toute information textuelle formulée en phrase et préférer un vocabulaire technique précis. Un "*ping timeout*" sera bien plus efficace que "*l'hôte n'a pas répondu à une requête de ping dans le délai attendu*". De même, l'utilisation de codes couleur pour exprimer le niveau de criticité de

13. Expression des Besoins et Identification des Objectifs de Sécurité [Colin, 2012]

l'état de santé est idéale. Nous savons que le rouge exprime aisément un problème critique, un danger, tandis que le vert exprime un état de bon fonctionnement.

D'autres part, l'iconographie utilisée doit être cohérente (i.e. niveau de détail, design, couleurs, etc.) au travers de tout le tableau de bord. Elle doit être explicite et favoriser la sémantique à l'esthétique.

L'administrateur doit pouvoir mémoriser les éléments visuels et textuels d'un tableau de bord. Leur nombre doit être limité et ils doivent être organisés suivant une structure hiérarchique cohérente. Il est, ainsi, préférable de limiter le nombre de modes de fonctionnement. Attention cependant à rester explicite et suffisamment précis. Réduire les modes de fonctionnement à "OK" et "NOK" est exagéré.

Empreinte de consommation minimale

La surveillance des systèmes impacte négativement leurs performances, mais elle est inévitable. Nous devons minimiser cet impact.

Quelle que soit la méthode employée, le système de surveillance requerra du système observé une part de ressources. Tout élément du système surveillant qui, par son fonctionnement, empêche la bonne exécution d'un processus de production est à proscrire. [Wikipedia, 2014e]

Le tableau de bord en lui-même ne consomme que les ressources de son hôte. Il devrait, idéalement, être hébergé sur une plateforme dédiée à la surveillance. Le système de surveillance doit faire l'objet d'une analyse et d'une mesure de performances avant d'être implémenté dans une infrastructure.

1.3 Standards et solutions professionnelles

Standards Internationaux

Ce sont principalement les recommandations de la série X.701-X.792 [14] de l'ITU [15] et l'ISO/IEC 10040 :1998 [16] qui définissent la gestion de systèmes d'information et leur surveillance. Ils décrivent notamment le protocole CMIP, ainsi que le modèle de gestion FCAPS (Fault, Configuration, Accounting, Performance, Security). FCAPS est un framework qui découpe en cinq catégories les grands axes de la gestion de réseaux informatique. Il est également repris dans les recommandations de l'ITU pour la gestion de réseaux de télécommunication. [Wikipedia, 2014b]

Le Distributed Management Task Force (DMTF) [17] se focalise particulièrement sur la gestion et l'administration des systèmes d'information. Le groupe est composé des plus grands noms de l'informatique : Cisco, Microsoft, Intel, Dell, VMWare, etc. Ils développent notamment les standars CIM [18] et WBEM [19] qui sont, par exemple, implémentés par Microsoft sous l'appelation connue "*Windows Management Instrumentation (WMI)*".

14. Le texte de ces recommandations est repris dans l'ISO/IEC 10164
15. International Telecommunication Union, `http://www.itu.int`
16. `http://www.iso.org/iso/home/store/catalogue\_tc/catalogue\_detail.htm?csnumber=24406`
17. DMTF `http://www.dmtf.org`
18. Common Information Model `http://www.dmtf.org/standards/cim`
19. Web-Based Enterprise Management `http://www.dmtf.org/standards/wbem`

Nous pouvons également citer les RFCs suivantes :

RFC1155	Structure and identification of management information for TCP/IP-based internets
RFC1157	Simple Network Management Protocol (SNMP)
RFC1213	Management Information Base for Network Management of TCP/IP-based internets :MIB-II

A la suite de nombreuses recherches infructueuses, nous décidons d'admettre qu'à l'heure actuelle, aucun standard international ne définit les tableaux de bord de gestion en eux-même. Il est cependant aisé, pour qui s'y intéresse, de trouver de nombreuses recommandations émises par l'industrie en terme de qualité de rapport et de présentation de l'information. Ceux-ci reprennent généralement les concepts émis à la section 1.2 de ce document mais s'intéressent peu à une représentation sous forme de graphes de dépendances.

Solutions existantes

Il existe énormément de solutions de surveillance informatique. Le tableau 1.1 liste quelques solutions parmi les plus populaires et abouties [20].

Logiciel	**Producteur**	**Description**
Nagios Core	Nagios Enterprises	Open-source en Python
Shinken	Jean Gabès	Fork de Nagios Core
Centreon	Merethis	Autre fork de Nagios
BMC	BMC Software	Suite complète de gestion d'infrastructure
SiteScope	HP	Membre de la suite BSM, gestion complète
PRTG	Paessler AG	Identification automatique des éléments
nVision	Axence	Principalement orienté réseau
Tivoli	IBM	Suite de solutions de gestion d'infrastructure
Applications Manager	ManageEngine	Solution complète

TABLE 1.1 – Solutions de surveillance informatique

Ces solutions intègrent, pour la plupart, des fonctionnalités qu'il est intéressant de mentionner :

- Détection des faux positifs : augmentation de la fréquence des contrôles lorsque l'un d'entre eux retourne une valeur considérée comme défaillance. L'alerte n'est générée qu'après un certain nombre de contrôles défaillants.
- Historique des valeurs de contrôle : présentation sous forme de graphique des valeurs des différents contrôles effectués sur une période prédéfinie. Les graphiques sont généralement générés à l'aide de l'outil RRDTools [21] et d'une base de données RRD.
- Détection de comportement inhabituel : sur base de l'historique de valeurs, le logiciel compare l'évolution de celles-ci à une période précédente (semaine, mois, année,...) et alerte lorsque les comportements ne sont pas similaires.

20. Une large liste comparative est disponible à l'adresse `http://en.wikipedia.org/wiki/Comparison_of_network_monitoring_systems`

21. Round-Robin Database Tools `http://oss.oetiker.ch/rrdtool/`

- Cartographie du réseau par un graphe : les connexions réseaux sont représentées à l'aide d'arêtes entre les différents éléments.
- Notion de dépendance : possibilité de définir une relation de dépendance entre plusieurs éléments/contrôles. Cela se limite généralement à réduire le nombre d'alertes causées par un même contrôle. Par exemple, si un switch tombe en panne, le système génèrera une alerte pour celui-ci, mais masquera celles qui concernent les serveurs qui y sont connectés puisqu'ils deviennent inaccessibles en raison du switch. Ainsi, l'administrateur ne rate pas l'alerte qui concerne le switch dans une masse d'alertes qui n'est que conséquence de la première.
- Accessibilité au travers d'un navigateur web : facilité d'accès et de contrôle du système de surveillance au travers d'une interface web.

Chapitre 2

Rappel de la théorie des graphes

La théorie des graphes s'applique à fournir une modélisation visuelle intuitive à la notion de relation binaire entre plusieurs éléments. Chaque élément est représenté par un point appelé *sommet* ou *nœud* et la relation est représentée par un *arc* orienté par une flèche. Cette représentation est ensuite exploitée pour extraire de nombreuses informations sur l'ensemble des relations. La notion de dépendance peut être réduite à un ensemble de relations binaires. Prenons le temps de rappeler les différents concepts et voyons comment ils s'appliquent dans notre cas. Nous nous concentrerons sur les notions qui nous concernent.

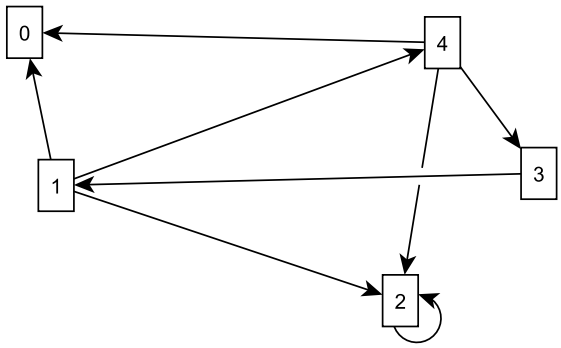

FIGURE 2.1 – Exemple de graphe orienté

2.1 Définitions [Leclerq, 2010]

Graphe : Un graphe G est la représentation d'une relation binaire sur une ensemble X, soit une partie du produit cartésien de X par lui-même ou encore un ensemble U de couples de X.

Nous noterons $G(X, U)$ un graphe dont l'ensemble des sommets est X et l'ensemble des couples, U. Un couple de U, donc une arc du graphe G, est noté (x, y). L'*ordre* du graphe est le nombre de sommets de X. La *taille* du graphe est le nombre d'arcs de U.

Un graphe est dit *orienté* si la relation qui lie chaque couple de sommets de U n'est pas symétrique. C'est le cas des relations de dépendance. En effet, le fait que A dépende de B n'implique pas que B dépende de A.

Un arc $u = (x, y)$ représente la relation de x vers y dans un graphe orienté. u est dit incident à y vers l'*intérieur* et incident à x vers l'*extérieur*.

Degré : le degré d'un sommet est le nombre total d'arcs incidents à ce sommet. Le *degré intérieur* est le nombre d'arcs incidents à ce sommet vers l'intérieur. Le *degré extérieur* est le nombre d'arcs incidents à l'extérieur.

Chemin : suite de n arcs $(x, y), (y, z), (z, t), ...$ tels que :
- l'extrémité du k^{eme} arc soit l'origine du $(k+1)^{eme}$ lorsque $n > 1$
- n est la longueur du chemin. Elle peut être nulle (chemin vide) ou égale à 1 (chemin réduit à un arc).

Un sommet i est dit *antécédent* d'un sommet j s'il existe un chemin de i vers j. j est alors appelé *descendant* de i

Circuit : chemin dont l'origine du premier arc coïncide avec l'extrémité du dernier. Un chemin ou un circuit est dit *élémentaire* s'il ne passe qu'une seule fois par chacun de ses sommets. Il est dit *simple* s'il ne contient qu'une seule fois chacun de ses arcs. Un circuit de longueur 1 ne contient qu'un seul sommet et un seul arc reliant celui-ci à lui-même. Un tel circuit est appelé *boucle*.

Dans un couple (x, y) d'un graphe orienté, le sommet x est le *précédent* de y, qui est lui-même le *suivant* de x. Les deux sommets sont dits *adjacents*. On note Γ la relation entre un sommet et l'ensemble de ses suivants. Elle est appelée *application multivoque*. L'*application réciproque* $\Gamma^{-1}(x) = \{y | (y, x) \in U\}$ est la relation entre x et l'ensemble de ses précédents.

La matrice A associée à un graphe G est une matrice booléenne de dimension $n \times n$ où n est la taille de G. Elle est définie par $A[i, j] = 1$ ssi $(i, j) \in U$. La matrice A associée au graphe de la figure 2.1 sera :

$$A = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \end{pmatrix}$$

Un graphe G contient une boucle si $\exists i : A[i, i] = 1$.

La matrice M carrée d'ordre n telle que $M_{ij} = 1$ s'il existe un chemin i vers j est dite *matrice d'accessibilité* ou *matrice de fermeture transitive*. La matrice M du graphe de la figure 2.1 est :

$$M = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Un graphe G contient un circuit de taille > 1 si $\exists i : A[i, i] = 0$ et $M[i, i] = 1$.

2.2 Propriétés et concepts [Leclerq, 2010]

Symétrie : Un graphe symétrique est une graphe dont la matrice associée est symétrique : $A[i,j] = A[j,i]$. Un graphe orienté symétrique pourrait être considéré comme un graphe non orienté sans perte d'information. Un graphe asymétrique est un graphe dont la matrice associée est asymétrique : elle vérifie $\forall A[i,j] + A[j,i] \leq 1$ c'est à dire $A[i,j] = 1 \Rightarrow A[j,i] = 0$. Un graphe asymétrique n'a pas de boucle.

Densité : la densité d'un graphe est le rapport entre le nombre m d'arcs et le nombre maximum possible $NMAX$ d'arcs. Dans un graphe orienté sans boucle d'ordre n, $NMAX = n(n-1)$.

Connexité forte : un graphe orienté est dit fortement connexe si pour tout couple de sommets x et y, il existe un chemin de x vers y et un chemin de y vers x. La matrice M d'un tel graphe est entièrement composée de 1. Un graphe fortement connexe contient forcément des circuits. Une composante fortement connexe d'un graphe G est un sous-graphe fortement connexe de G.

Pondération : un *poid* peut être attribué à chaque sommet ou arc. Ce poid représente une valeur quelconque déterminée par ce que représente le graphe. La pondération est utilisée dans des cas de calcul d'optimisation.

2.3 Graphes de dépendances

Les graphes de dépendances sont des graphes orientés spécifiquement utilisés pour exprimer la notion de dépendance d'un sommet par rapport à un autre. Ils sont notamment utilisés dans l'ingénierie du logiciel où ils permettent de représenter les dépendances entre plusieurs packages ou services liées aux appels de fonction ou à l'utilisation de données. De nombreuses études existent sur le sujet : l'étude de Zimmermann et Nagappan [2008] décrit l'exploitation de ces graphes dans le cadre d'une prédiction de défaillance d'un composant sur base de sa dépendance à d'autres [Zimmermann et Nagappan, 2008]. Ils sont également utilisés dans le traitement du langage naturel en introduisant une notion de dépendance entre deux mots d'une même phrase [Nivre et McDonald, 2008].

Les graphes de dépendances sont généralement acycliques car un circuit dénote un problème fondamental au niveau du design du système. Pour cette raison, il est de notre opinion que la présence de circuits, bien que possible, devrait générer une alerte. Cette opinion est partagée par Alexander Keller et Gautam Kar, chercheurs au centre IBM T.J. Watson Research Center.

> « *The dependency graph of services is directed and acyclic : The latter statement reflects the authors' experience with IP-based networked services. It is the authors' belief that while such configuration is technically possible, it reflects flaws in the system design because this leads to an unstable system [...]. A dependency-checking application that discovers cyclic dependencies should issue a warning to an administrator.* »
> [Keller et Kar, 2001]

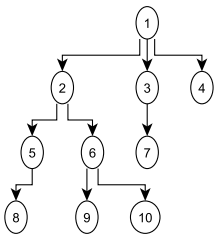

FIGURE 2.2 – Ordre de parcours par BFS

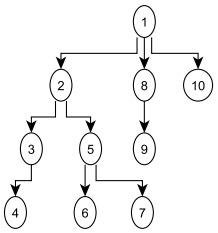

FIGURE 2.3 – Ordre de parcours par DFS

2.4 Algorithmes

Les graphes de dépendances que nous exploiteront dans cette étude ne sont pas pondérés, nous nous concentrons donc sur les algorithmes de manipulation des graphes non-pondérés.

2.4.1 Breadth First Search et Depth First Search

Breadth First Search (BFS) et Depth First Search (DFS) sont des algorithmes servant à parcourir le graphe au départ d'un sommet donné. Ils sont utilisés pour déterminer l'existence de chemins entre deux nœuds, vérifier la présence de circuits ou encore déterminer la connexité du graphe.

Le premier, BFS, met en priorité l'analyse de tous les suivants directs d'un nœud avant de passer aux autres descendants (parcours en largeur). DFS quant à lui, privilégie les descendants (parcours en profondeur). Les figures 2.2 et 2.3 présentent respectivement l'ordre de parcours de BFS et DFS.

2.4.2 Détection de circuit

Si DFS est capable de détecter la présence d'un circuit, il est limité à l'identification de deux des nœuds consécutifs de ce circuit. Il existe d'autres algorithmes capables de retrouver les composantes fortement connexes d'un graphe, et donc ses circuits. Notamment, Tarjan's strongly connected component [1] et Malgrange.

2.5 Structures de données

Comme pour toutes les données, nous disposons de plusieurs structures pour les manipuler dans un logiciel. Afin de limiter leur nombre et ne garder que les plus efficaces, il est essentiel de déterminer de quelles informations nous devons disposer suivant l'utilisation que nous en ferons. Pour chaque nœud, nous avons besoin de répondre aux questions suivantes :

– Quelle est sa relation avec le sommet i ? Est-il suivant, précédent, antécédent, descendant ?

1. `http://en.algoritmy.net/article/44220/Tarjans-algorithm`

– Quels sont tous les suivants, précédents de ce sommet ?
– Quel est le degré intérieur et extérieur du sommet ?

Mathématiquement, la représentation d'un graphe par ses matrices d'adjacence et d'accessibilité est la plus efficace puisque la plupart des informations peuvent être déduites par calcul matriciel [Leclerq, 2010]. Cependant, d'un point de vue informatique, ce n'est pas la plus économique. En effet, un graphe dont la densité est faible nécessitera autant d'espace mémoire qu'un graphe de même ordre mais plus dense. De plus, pour analyser le graphe, il faudra parcourir l'entièreté des matrices. Ce qui représente un rendement très faible pour les graphes peu denses.

D'une manière générale, les solutions de manipulation de graphes actuelles utilisent la conception orienté-objet. Les sommets sont instanciés à partir d'une classe "*sommet*", le graphe est une collection d'objet de cette classe. Suivant le contexte d'utilisation du graphe, il est opportun de créer une classe "relation" qui représente la relation entre deux sommets — notamment lorsque la relation doit contenir plus d'informations (e.g. pondération) ou simplement d'associer à chaque sommet une liste de ses précédents directs et une liste de ses suivants directs. La figure 2.4 présente les deux cas. Chaque sommet peut également contenir la liste des relations qui le concernent. Suivant les cas d'utilisation, les listes et collections d'objet peuvent être implémentées différemment (i.e. triées ou non, indexation, table de hachage, etc.)

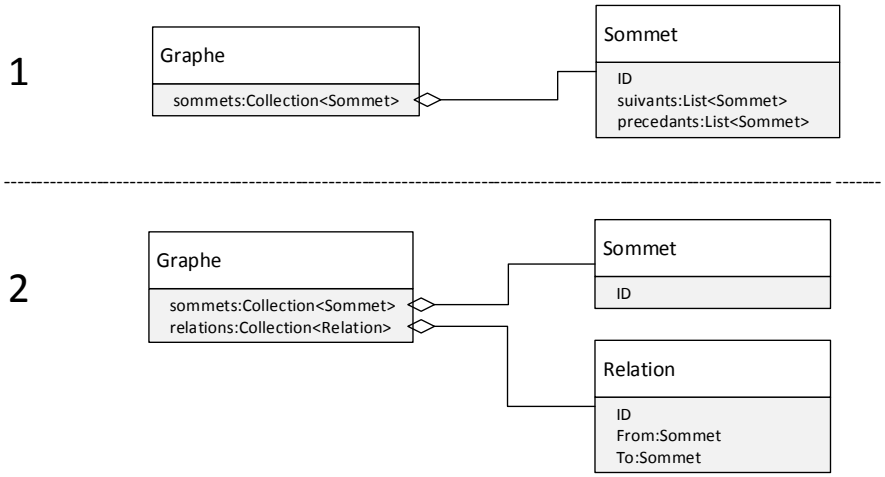

FIGURE 2.4 – Représentation d'un graphe en Orienté-Objet

Notons que les sociétés yWorks [2] et Cytoscape Consortium [3] sont particulièrement avancées dans ce domaine et fournissent des librairies pour la manipulation de toutes sortes de graphes pour différentes plateformes et langages (e.g. Java, Web, Android,...).

2. http://www.yworks.com
3. http://www.cytoscapeconsortium.org/

2.5.1 Formats de fichier

Si la structure des données pour la manipulation du graphe est cruciale, il en est tout autant pour le stockage de ce dernier entre deux manipulations. Le format de fichier est également important pour le dialogue entre plusieurs applications. A cet effet, il existe plusieurs standards et formats ouverts et documentés. L'annexe A présente différents exemples pour ces formats.

Graph Modelling Language - GML
Egalement appelé *Graph Meta Language*, GML est développé par l'Université de Passau [4] (Allemagne). Il est portable, extensible et flexible. Il est essentiellement basé sur une structure hiérarchique d'une liste de type clé-valeur. [Himsolt, 2010]

Graph eXchange Language - GXL [Holt *et al.*, 2002]
GXL [5] est un langage développé comme un format d'échange standard pour les graphes, par Ric Holt , Andy Schürr, Susan Elliott Sim et Andreas Winter, chercheurs dans différentes universités. Ce langage est suivi par plusieurs industries informatiques (e.g. Bell Canada, IBM, Nokia, Philips) et universités (e.g. Berlin, Berne, Edinburg, Oregon, Toronto). Il est notamment inspiré de GML.

eXtensible Graph Markup and Modeling Language - XGMML
XGMML [6] est une transposition directe de GML au format XML. L'objectif était l'utilisation de ce format dans le cadre du système WWWPAL [7] qui représente les documents web par un graphe.

GraphML
Le format GraphML [8] est développé par le Graph Drawing Steering Committee et supporte tout type de graphe. Il exploite essentiellement le format XML et s'inspire également de GML. Sa portabilité et sa simplicité en font un bon candidat pour de nombreuses applications. Il existe une version compressée de ce format : Compressed GraphML (*.graphmlz*). Aucune information n'est perdue par cette compression.

Graphviz et le langage DOT
Graphviz [9] est un format développé par AT&T et utilise le langage DOT [10], entièrement dédié aux graphes. Le langage DOT étant fort proche de GML, la transition est aisée entre ces deux formats (e.g. commande *gml2gv* sur linux).

4. `http://www.uni-passau.de/en/`
5. `http://www.gupro.de/GXL`
6. `http://cgi7.cs.rpi.edu/research/groups/pb/punin/public_html/XGMML/`
7. `http://www.cs.rpi.edu/~moorthy/AACE/wwwpal_paper.html`
8. `http://graphml.graphdrawing.org`
9. `http://graphviz.org`
10. Egalement développé par AT&T `http://www.graphviz.org/content/dot-language`

2.5.2 Bases de données

De nombreux systèmes de gestion de bases de données (SGBD) se sont focalisés sur le stockage de graphes et les langages de requêtes. Le site internet DB-engines.com établit un classement [11] de l'utilisation des SGBD. La table 2.1 présente le classement des SGBD pour graphes [12] pour le mois de juin 2014.

Rank	Last Month	DBMS	Database Model	Score	Changes
1.	1.	Neo4j	Graph DBMS	22.50	+1.04
2.	2.	Titan	Graph DBMS	1.86	−0.01
3.	3.	OrientDB	Multi-model	1.73	+0.05
4.	4.	Sparksee	Graph DBMS	0.82	+0.02
5.	5.	Giraph	Graph DBMS	0.41	+0.04
6.	9.	ArangoDB	Multi-model	0.20	+0.13
7.	6.	InfiniteGraph	Graph DBMS	0.15	+0.02
8.	7.	InfoGrid	Graph DBMS	0.09	+0.01
9.	8.	FlockDB	Graph DBMS	0.08	+0.00
10.	10.	Sqrrl	Multi-model	0.04	+0.01
11.	11.	Amisa Server	Multi-model	0.00	±0.00
11.	11.	GlobalsDB	Multi-model	0.00	±0.00
11.	11.	GraphBase	Graph DBMS	0.00	±0.00
11.	11.	HyperGraphDB	Graph DBMS	0.00	±0.00

TABLE 2.1 – Classement des SGBD pour graphes de DB-engines.com, Juin 2014

Neo4j
Neo4j, de loin le plus utilisé actuellement, est développé par la société Neo Technology. Il est Open Source [13], sous licence GPLv3 (Community edition) et AGPLv3 (Enterprise edition) pour les projets Open Source, sous licence commerciale pour les autres. Il est relativement simple à mettre en place et dispose d'une documentation complète et d'une communauté active.

Il est développé en Scala et propose des interfaces REST et HTTP, le rendant accessible par la plupart des langages applicatifs. Il est conforme aux propriétés ACID [14] garantissant l'intégrité des données.

Une base de données contient un seul graphe qui peut représenter plusieurs types de relations simultanément. Les nœuds et les relations peuvent contenir plusieurs arguments textuels arbitraires.

Neo Technology développe également le langage de requête déclaratif Open Source pour graphe Cypher [15], exploité dans Neo4j, qui permet de manipuler les nœuds et leurs relations.

11. Plus d'informations sur les critères de classement disponibles à l'adresse `http://db-engines.com/en/ranking_definition`
12. `http://db-engines.com/en/ranking/graph+dbms`
13. `https://github.com/neo4j/neo4j`
14. Atomicity, Consistency, Isolation, Durability
15. Plus d'informations à l'adresse `http://neo4j.com/guides/basic-cypher/`

Neo4j est également compatible avec le langage de requête Open Source Gremlin [16], plus complet que Cypher. Cependant, ce dernier semble avoir une courbe d'évolution plus importante, du fait de sa simplicité intuitive et de la popularité de Neo4j.

Neo4j rejoint la famille de SGBD appelée "*NoSQL*" (Not only SQL), très présente dans la tendance actuelle de *BigData*.

16. `https://github.com/tinkerpop/gremlin/wiki`

Chapitre 3

Méthodes d'analyse et recherches liées

Il existe beaucoup de méthodes d'analyse dans le domaine de la fiabilité et de la gestion des systèmes d'information. Au travers de nos recherches nous avons également pu découvrir les résultats obtenus par plusieurs chercheurs qui se sont intéressés directement ou indirectement à notre problématique.

3.1 Méthodes d'analyses

3.1.1 Impact Analysis (IA)

L'analyse d'impact consiste à déterminer dans quelle mesure un évènement donné influencera le comportement, l'évolution d'un groupe d'éléments déterminé. Cet évènement peut aller du changement d'un ou plusieurs éléments à la suppression ou l'ajout de ceux-ci, en passant par leur défaillance.

Elle prend différentes formes suivant le type d'utilisation. On parle souvent de Change Impact Analysis, Business Impact Analysis, Test Impact Analysis, et bien d'autres.

Le principe général est de déterminer un évènement à analyser, le type d'impact que l'on veut observer et de construire un graphe d'impact représentant l'enchaînement de conséquences qu'aura l'évènement initial. L'analyse quantifie finalement l'impact à partir du graphe.

Dans le cadre des graphes de dépendances, nous pouvons aisément construire le graphe d'impact. En effet, si l'élément A dépend de l'élément B, alors un changement opéré sur B aura un impact sur A. Le graphe d'impact est simplement identique au graphe de dépendance mais parcouru en sens inverse. L'analyse d'impact revient alors à quantifier les dépendances en terme de valeur d'impact suivant les différents évènements intéressants, et de dresser une liste de celles-ci pour un évènement donné.

3.1.2 Root Cause Analysis (RCA)

L'analyse de causes racines, à l'inverse, détermine le ou les évènements dont la chaîne causale aboutit à un évènement donné. Il s'agit ici de construire un graphe de type arborescence dans lequel chaque évènement est décomposé en causes possibles. Chaque nouvelle cause découverte est analysée jusqu'à trouver l'évènement déclencheur.

L'analyste dresse ensuite la liste des causes racines et s'attèle à définir des actions correctives pour chacune d'entre elles.

Dans notre cas d'étude, elle présente deux aspects que nous détaillerons au chapitre 6 :

- Premièrement, il est question de déterminer quelles suites de liens de dépendance ont conduit à l'état de santé d'un élément
- Deuxièmement, il faut réaliser un travail d'investigation pour déterminer pourquoi un élément est annoncé défaillant par le système de surveillance et donc la cause de cette défaillance

3.1.3 FTA, FMEA, FMECA, DCCA, CFIA

Les études suivantes sont essentiellement prévues pour la phase initiale des projets. Elles permettent de mettre à l'épreuve le design élaboré pour un produit. Nous survolerons ces méthodes car nous nous concentrons sur l'exécution du système d'information et non sa conception. Cependant, il est intéressant de les mentionner pour ouvrir des pistes d'évolution future.

FTA [USNRC, 1998]
Fault Tree Analysis, à l'instar de RCA, se focalise sur un évènement et en détermine les causes. Ce pendant, elle ne cherche pas les causes réelles, mais toutes les causes possibles et probables.

L'analyse construit un arbre de causes dont l'évènement analysé est la racine. Chaque nœud est décomposé en sous-nœuds reliés par des nœuds *ET* et *OU*, à l'image des portes logiques. La taille de l'arbre dépend du niveau de détail que l'on souhaite atteindre.

Pour chaque feuille de l'arbre, l'analyste détermine une probabilité d'occurrence. Celles-ci sont ensuite combinées pour calculer la probabilité du nœud parent en fonction des portes *ET* et *OU*. La probabilité d'occurrence de l'évènement analysé est ainsi déterminée.

FMEA [Rizk et Ratajczak, 2012] Failure Mode and Effect Analysis est une analyse qui s'attèle à lister les étapes de processus d'exécution et d'en définir les potentiels modes de dysfonctionnement en y associant la sévérité, les causes possibles et les actions correctives recommandées. A chaque mode est associé la sévérité, la fréquence d'apparition et les chances de détection lors de la phase de test. Les échelles de ces valeurs sont définies par l'analyste suivant les besoins de l'analyse. Il est possible d'utiliser l'analyse FTA pour déterminer ces valeurs. Nous donnons, a titre d'exemple, des échelles dans les tables 3.1, 3.2 et 3.3.

1	Aucun effet notable
2	Effets très mineurs, aucun dommage
3	Effets mineurs, dommages légers
4	Effets modérés, dommages modérés
5	Effets critiques (perte de fonction primaire, perte de marge de sécurité, etc.)
6	Effets catastrophiques (arrêt de l'appareil)

TABLE 3.1 – Echelle de sévérité

1	Très peu probable
2	Faible
3	Occasionnel
4	Raisonnablement possible
5	Fréquent

TABLE 3.2 – Echelle de fréquence d'apparition

1	Certaine
2	Presque certaine
3	Elevée
4	Modérée
5	Faible
6	Non détecté

TABLE 3.3 – Echelle de chances de détection

L'ensemble de l'analyse est présentée sous forme de tableau reprenant, en ligne, les différents modes de dysfonctionnement, et en colonne les différentes données de chacun.

FMECA [Robert Borgovini et Rossi, 1993] Failure Mode, Effects and Criticality Analysis est une analyse qui complète FMEA par l'ajout de la notion de criticité. Celle-ci est déterminée par une valeur appellée "Risk Priority Number" (RPN) et établi une échelle de priorité de correction. Le RPN est calculé sur base de la sévérité, de la fréquence d'apparition et les chances de détection.

DCCA [Ortmeier *et al.*, 2006] Deductive Cause-Consequence Analysis est une généralisation formelle de FMEA et FTA dans laquelle les raisonnements informels sont remplacés par des analyses et de preuves mathématiques. Celles-ci sont réalisées à l'aide d'une modélisation par automates finis et d'une logique de calcul en arbre (Computation Tree Logic)

CFIA Component Failure Impact Analysis présente sous forme de tableau l'ensemble des composants d'un système et détermine le niveau d'impact de leur arrêt sur les utilisateurs du

système, sur base d'une échelle simplifiée. Les modes de dysfonctionnement et leurs effets sont également détaillés.

Cette analyse se rapproche de près à notre problématique mais se limite à établir un constat et alimenter une documentation. Elle ne s'intéresse pas à l'état de fonctionnement en temps réel ni aux relations entre les composants.

3.2 Recherches liées

3.2.1 Keller et Kar, 2001

Les travaux de Keller et Kar sont particulièrement intéressants pour notre solution. Dans leur article *Determining Service Dependencies in Distributed Systems* [Keller et Kar, 2001], ils proposent une architecture et son implémentation pour identifier automatiquement les dépendances techniques entre les services d'un environnement hétéroclite de systèmes d'information. Nous exploiterons le potentiel de cette étude dans le chapitre 7.

L'étude présente les différents services sous forme d'un graphe de dépendances. Le choix de cette représentation vient de l'identification de deux objectifs qui sont fondamentalement identiques à ceux du présent document :

1. D'une part, lors d'une défaillance, l'identification des causes probables du problème peut se faire à l'aide d' Analyse de causes racines (Root Cause Analysis, voir sous-section 3.1.2). Il est alors question de parcourir l'arbre de dépendances dont le nœud défaillant est la racine.
2. D'autre part, lors d'une maintenance importante, il est essentiel pour les administrateurs de pouvoir déterminer à l'avance quels services seront impactés. Il est, cette fois, question de parcourir le graphe de dépendances dans le sens inverse en partant du nœud concerné par la maintenance et d'effectuer ainsi une analyse d'impact (Impact Analysis, voir sous-section 3.1.1).

Cette architecture se base sur l'hypothèse que chaque service et chaque hôte est capable de fournir une identification de leur système et de leur différentes dépendances au format XML, ces fichiers sont supposés maintenus à jour par les services. Si cela n'est pas le cas pour tous les équipements et services, il est aisé d'implémenter cette même fonctionnalité à l'aide d'agents logiciels, notamment en se basant sur une architecture proposée par Kar et al. dans l'article *Managing Application Services over Service Provider Networks : Architecture and Dependency Analysis* [Kar *et al.*, 2000] décrite à la section 3.2.2.

L'étude identifie également trois modèles de dépendance liés au cycle de vie d'un service à des niveaux de granularité différents (Fig. 3.1).

Principe
La proposition est basée sur une architecture trois tiers et suit une approche *Diviser pour régner* en raison du nombre potentiellement élevé de services et pour répondre à la problématique d'évolutivité dans les environnements distribués. Le composant central de cette architecture est appelé *Dependency Query Facility* (DQF). Le principe est le suivant :

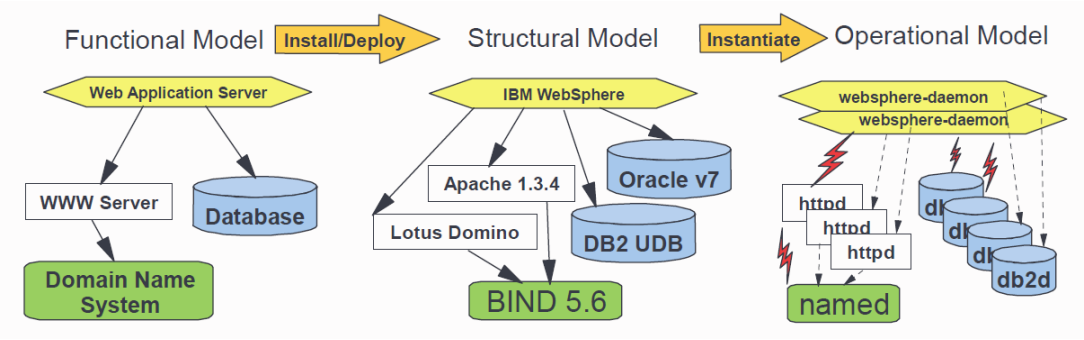

FIGURE 3.1 – Dependency models related to the service lifecycle [Keller et Kar, 2001, Fig 1]

1. Le DQF interagit avec le système de gestion (à gauche sur la figure 3.2). Il reçoit les requêtes au travers d'une API qui expose des fonctionnalités flexibles de "*drill-down*" et "*drill-up*" permettant, par exemple, de retrouver les antécédents et précédents d'un nœud, un sous-graphe partant d'un nœud, ou encore un sous ensemble arbitraire.
2. Le DQF récupère les informations de dépendance des différents services au travers d'un protocole HTTP dans un dépôt où sont stockés les fichiers XML des services (à droite).
3. Le DQF applique les critères de la requête sur les informations compilées et retourne le résultat au système de gestion dans un format XML prédéfini.

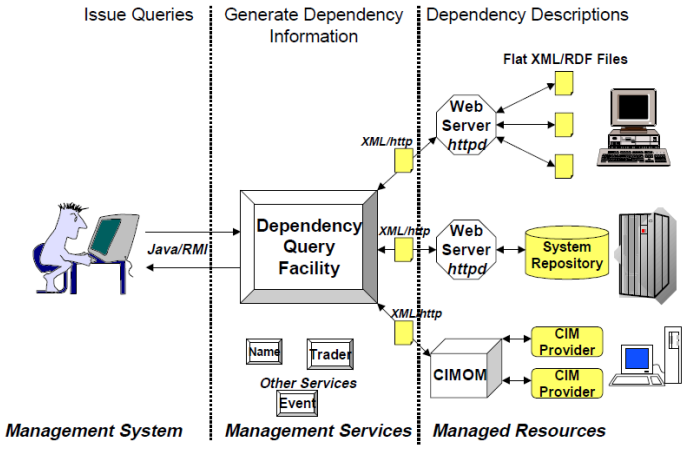

FIGURE 3.2 – Architecture of the Dependency System [Keller et Kar, 2001, Fig 2]

Le prototype d'implémentation de Keller et Kar présente un cas d'utilisation dans un environnement e-commerce constitué d'un serveur Windows NT4 hébergeant l'application et le serveur web, et un serveur AIX hébergeant la base de données (Fig. 3.3).

Chaque élément du graphe, aussi bien les services que leur dépendances et leurs attributs respectifs, sont représentés comme des éléments du document XML. L'approche de Keller et Kar est la suivante :

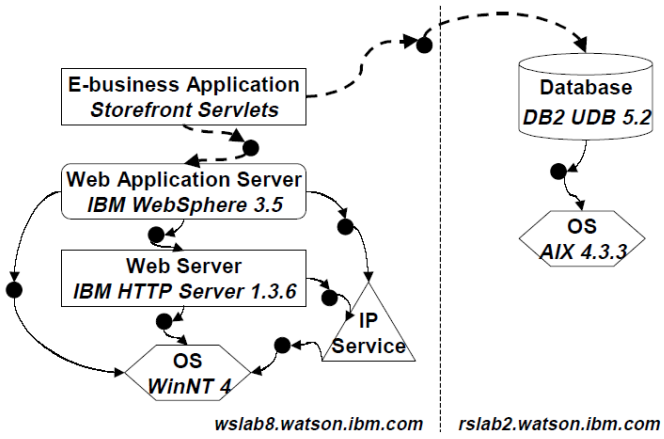

FIGURE 3.3 – Sample service dependency graph [Keller et Kar, 2001, Fig 3]

1. Exploiter les possibilités de *parsing* de XML pour représenter les informations de dépendance.
2. Exploiter le Resource Description Framework (RDF [1]) pour représenter le graphe de dépendances au moyen d'une approche hybride RDF/XML (voir l'exemple de fichier XML ci-dessous).
3. Exploiter les facilités de requêtes du XML Path Language (XPath) pour sélectionner et filtrer les objets.

L'exemple ci-dessous présente ce que contiendrait le fichier XML généré par le service *E-Business Application Storefront Servlets* de la figure 3.3. Nous y retrouvons les deux antécédents du service aux lignes 15 et 23. Ces antécédents contiennent l'URI de leur fichier XML respectifs, décrits dans une ressource RDF.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <rdf:RDF xmlns:ds="wslab4/DependencySchema#"
3 xmlns:rdf="www.w3.org/1999/02/22-rdf-syntax-ns#"
4 xmlns:rdfs="www.w3.org/2000/01/rdf-schema#">
5   <ds:Service>
6     <ds:name>E-business Application</ds:name>
7     <ds:caption>Storefront Servlets</ds:caption>
8     <ds:identifier>catalogServlets</ds:identifier>
9     <ds:description>myCatalogApp</ds:description>
10    <ds:version>3</ds:version>
11    <ds:release>1</ds:release>
12    <ds:processName></ds:processName>
13    <ds:dependency>
14      <ds:ServiceDependency>
15        <ds:antecedent rdf:resource=
16             "http://rslab2/xmlrepos/db2.xml"/>
```

1. RDF est un format standardisé par le W3C `http://www.w3.org/RDF/`

```
17        <ds:generated>automatic</ds:generated>
18        <ds:label>App DB Dependency</ds:label>
19      </ds:ServiceDependency>
20    </ds:dependency>
21    <ds:dependency>
22      <ds:ServiceDependency>
23        <ds:antecedent rdf:resource=
24             "http://wslab8/xmlrepos/websph35.xml"/>
25        <ds:generated>automatic</ds:generated>
26        <ds:label>App AppServer Dependency</ds:label>
27      </ds:ServiceDependency>
28    </ds:dependency>
29  </ds:Service>
30 </rdf:RDF>
```

Dans l'exemple, la requête XPath "*/descendant : :ds :Service[@rds :about=ID]*" permettrait de chercher, parmi tous les nœuds descendants du nœud courant ("*/descendant*"), les nœuds "*ds :Service*" qui auraient un attribut "*rdf :about*" dont la valeur serait "*ID*".

3.2.2 Kar et al., 2000

L'étude de Kar et al. intitulée *Managing Application Services over Service Provider Networks : Architecture and Dependency Analysis* [Kar *et al.*, 2000], dans sa section 4, expose une solution afin de déterminer un certain nombre de dépendances entre plusieurs services intra- et inter-systèmes. Cette solution se base sur l'affirmation que, au sein de la plupart des systèmes d'exploitation (SE) actuels, se trouve une base d'information sur les applications qui sont installées. C'est effectivement le cas pour les systèmes Windows (e.g. WMI, base de registre) et UNIX (e.g. AIX Object Data Manager pour IBM AIX, Red Hat Package Manager (RPM) pour Linux). Ces bases d'informations peuvent être consultées à l'aide de scripts et de commandes et donc il est possible d'automatiser leur analyse. Cette approche est appelée "*Static Dependency Analysis*".

Dans le cas des dépendances intra-système, la solution exploite principalement les informations de pré-requis d'installation pour déterminer la dépendance d'un service par rapport à un autre. La section 4.2 de l'étude présente un exemple sur base d'AIX ODM.

En ce qui concerne les dépendances inter-système, la solution exploite cette fois les informations de configuration des applications clientes installées et présentes dans les bases d'informations accessibles. En effet, si un service contient dans sa configuration un partage NFS, nous pouvons déduire que ce service est dépendant d'un client NFS fonctionnel, du serveur proposant le partage NFS défini dans la configuration, d'un service DNS (probablement présent à un autre endroit de la configuration du SE), et d'une connectivité TCP-IP fonctionnelle.

Bien entendu, toutes les dépendances d'un système d'information distribué ne peuvent pas être déduites de cette façon. Cependant cela permet d'identifier la plupart des dépendances techniques.

3.2.3 Autres études

D'autres études méritent d'être mentionnées dans le présent document étant donné leur utilité dans la gestion d'incidents dans les systèmes d'information distribués à l'aide de graphe de dépendances entre services.

Notamment, l'étude de Ensel, 1999, intitulée *Automated generation of dependency models for service management* [Ensel, 1999] suggère l'utilisation de réseaux de neurones pour déterminer des dépendances entre deux services sur base de leur utilisation simultanée de ressources système.

L'étude de Bagchi et al., 2001, intitulée *Dependency Analysis in Distributed Systems using Fault Injection : Application to Problem Determination in an e-commerce Environment* [Bagchi *et al.*, 2001] propose une approche appelée *Active Dependency Discovery* et consiste à perturber un système en y injectant un dysfonctionnement et observer le comportement des systèmes environnants. Notons qu'une telle approche est à éviter dans un environnement de production. Cette étude se base sur celle de Brown et al.,2001, *"An Active Approach to Characterizing Dynamic Dependencies for Problem Determination in a Distributed Application Environment"* [Brown *et al.*, 2001] qui propose la même approche.

Bodenstaff et al., dans leur article *Monitoring Dependencies for SLAs : The MoDe4SLA Approach* [Bodenstaff *et al.*, 2008] proposent une approche exploitant les graphes de dépendances pour gérer les dépendances entre SLAs [2].

Une autre étude souvent citée dans la littérature est celle de Gruschke et al. 1998 *Integrated event management : Event correlation using dependency graphs* [Gruschke *et al.*, 1998] qui décrit l'utilisation des mêmes graphes de dépendance pour la corrélation d'évènements.

2. Service Level Agreements

Deuxième partie

Conception de la solution

Chapitre 4

Utilité et description

Nous avons posé et défini les bases de la problématique. Afin de répondre efficacement à celle-ci, il nous a fallu faire des choix et réfléchir à l'utilité de concevoir une solution logicielle adéquate. Nous détaillons maintenant le fruit de notre réflexion.

4.1 Modélisation sous forme de graphes de dépendances

L'une des contraintes de la présente étude est la nécessité d'exprimer et représenter les dépendances entre les différents éléments d'une salle machine. Ce n'est donc pas un choix de l'auteur. Cependant nous adhérons parfaitement à cette contrainte tant l'utilité d'une telle représentation, décrite ci-dessous, est importante. Le choix d'exploiter la théorie des graphes, et particulièrement les graphes de dépendances vient naturellement dans ce cas de figure.

Nous savons d'expérience que le cycle de vie d'une infrastructure informatique inclut rarement plus d'une phase de conception globale. Même lors d'un déménagement ou d'un rachat de l'entreprise, l'infrastructure est rarement recréée à partir de rien. Certaines parties sont modifiées, améliorées, voire parfois remplacées ou même supprimées, mais l'infrastructure n'est pas remise à zéro ; elle évolue. De plus, elle est généralement un ensemble très hétéroclite aussi bien en terme de services instanciés, que de systèmes d'exploitation et même de fabricants de matériels et d'années de fabrication. C'est pourquoi nous élaborons notre solution avec l'objectif de pouvoir l'implémenter et l'intégrer dans un quelconque environnement existant, avec un minimum d'effort et une compatibilité maximisée. Nous décrirons dans le chapitre 7 comment et par quels moyens effectuer cette intégration ainsi que différentes techniques pour automatiser la génération de la représentation. Néanmoins, l'architecture proposée peut, sans plus de difficulté, être implémentée à la naissance de l'infrastructure et suivre son évolution. Comme le dit l'adage, *qui peut le plus, peut le moins* [Aristote].

Le tableau de bord tel que nous le concevons s'adresse à deux fonctions dans l'entreprise : l'administrateur système et le manager informatique. Par extension, il pourrait également s'adresser aux opérateurs.

4.1.1 Utilité pour l'administration des systèmes

L'exercice de réaliser le graphe de dépendances d'une infrastructure informatique a plusieurs avantages pour l'administrateur. D'une part il permet d'avoir une meilleure connaissance de l'infrastructure, d'avoir une vision plus globale et donc de pouvoir exercer une meilleure gestion de celle-ci.

Ensuite, cela permet d'exposer des zones oubliées ou inconnues de l'infrastructure. Il n'est pas rare d'oublier un petit système considéré fonctionnel, installé lors d'un test ou encore supposé temporaire mais qui consomme trop de ressources et pourrait être optimisé, remplacé ou supprimé. Les administrateurs en charge d'infrastructures multiples et complexes ont parfois tendance à mettre en place des solutions de secours qui se fondent dans la masse de systèmes. Ils peuvent entraîner des conséquences néfastes s'ils ne sont pas pris en compte lors de phases d'évolution de l'infrastructure. L'exercice est donc l'occasion de revenir sur ces systèmes oubliés.

Un autre point non négligeable est le fait que, inévitablement, l'infrastructure contiendra des équipements qui ne peuvent pas être surveillés automatiquement. Soit en raison de leur ancienneté, soit simplement parce qu'ils ne fournissent pas d'interfaces exploitables. Ils peuvent avoir un impact sur les autres systèmes et un système de surveillance classique ne sera pas suffisant pour les considérer. L'intérêt ici est de pouvoir les mettre en relation dans une représentation qui permet de les inclure dans une analyse de causes racines et d'impact d'un autre composant.

Un dernier avantage essentiel est la mise en évidence de sections critiques de l'infrastructure et des défauts de conception. Un élément dont dépendent beaucoup de systèmes devrait mériter qu'on s'intéresse à sa haute disponibilité. De même, nous pouvons établir des comparaisons entre l'infrastructure et les bonnes pratiques publiées par l'industrie.

Le tableau de bord est également une excellente source d'information et de documentation et permet à l'administrateur de ne pas devoir mémoriser l'entièreté de l'infrastructure.

4.1.2 Utilité décisionnelle

Comme nous l'avons décrit dans la section 1.2, une décision correcte nécessite une base d'information. Une présentation des dépendances au sein du système d'information est idéale pour les décisions que doivent prendre les managers.

Nous l'avons précisé dans les sections 3.1.1 et 3.1.2, les résultats des analyses d'impact et de causes racines sont primordiaux pour des décisions éclairées. Nous pouvons également déduire plusieurs utilités décisionnelles de la section 4.1.1.

Un tableau de bord est précieux pour l'établissement de priorités dans les projets qui touchent à l'infrastructure.

4.2 Représentation par superposition de calques

Une telle structure est idéale pour représenter tous les aspects de l'administration système. Chaque calque représente un et un seul type de relation bien précis, un aspect de l'infrastructure. C'est fondamentalement une transposition directe de la vision qu'ont les administrateurs. Il leur est donc intuitif de la manipuler. Nous pouvons ensuite exploiter l'ensemble des relations décrites dans les calques pour en déduire le graphe de dépendances complet de l'infrastructure. Le fait d'afficher ou masquer un calque permet de personnaliser la vue que l'on souhaite avoir de l'infrastructure. La figure 4.1 présente la structure du tableau de bord.

Pour que le graphe de dépendances global soit efficace, toutes les relations qui induisent de nouvelles dépendances (hébergement, communication) doivent obligatoirement être présentes. La distinction en calque est nécessaire car nous souhaitons exprimer des relations différentes entre les composants. Nous ne pouvons pas, dans un même graphe, exprimer plusieurs types de relations (e.g. relation de dépendance et relation de communication).

Nous identifions trois types de calque qui sont indispensables aux objectifs de la présente étude : les relations de communication, les relations d'hébergement et les relations de dépendance.

La représentation détaillée des éléments de ces calques est décrite dans le chapitre 5.

4.2.1 Calque d'hébergement

Le calque d'hébergement représente la relation "A est hébergé par l'hôte B". Ces relations impliquent la dépendance directe "A dépend de B". Tout service logiciel est hébergé par un ou plusieurs matériels (e.g. serveur, modem, SAN, etc.).

C'est donc dans ce calque que nous retrouverons les liens entre services, systèmes d'exploitation, et matériels ainsi que les différentes couches de virtualisation.

Ce calque peut être alimenté par exploration du système distribué (voir chapitre 7).

4.2.2 Calques de communication

Si l'élément A dépend de C, c'est forcément qu'ils communiquent d'une façon ou d'une autre, qu'ils sont inter-connectés. Que ce soit une communication TCP-IP, un lien SCSI, un échange de données dans un flux d'informations ou même un transfert physique manuel (e.g. déplacement d'une bande de backup), il doit exister entre deux éléments dépendants une communication uni- ou bidirectionnelle. Si cette-ci est unidirectionnelle, elle doit, bien évidemment, être dans la même direction que la dépendance.

Autrement dit, si chaque calque de communication contient un type de relation, et que ces relations sont représentées sous forme de graphes (orientés ou non), il doit être possible dans l'un d'entre eux de trouver un chemin par lequel B est accessible pour A. Puisque nous manipulons ici des environnements informatiques distribués, nous pouvons assumer que le calque de communication TCP-IP sera présent.

De ces communications, nous pouvons déduire que l'élément A, en plus de dépendre de C, dépend également de tous les nœuds qui composent le chemin de communication entre A et C.

Notons que le chemin en question peut transiter par la relation d'hébergement. La communication entre l'hôte de A et l'hôte de B est généralement suffisante pour que A et B puissent interagir. Notons également que la relation de dépendance doit comporter l'information indiquant quel(s) type(s) de communication la supporte(nt).

4.2.3 Calque de dépendances

Ce calque représente toutes les dépendances qui ne peuvent pas être déduites à partir des autres calques. Par exemple, la relation "Service a dépend du système d'exploitation b" peut être déduite à partir de la relation "Sa est hébergé par SEb" du calque d'hébergement. Il est de notre avis que la représentation d'une telle dépendance dans ce calque nuit à la lecture du graphe représenté. Cette dépendance est suffisamment implicite et maitrisée par les administrateurs. Le graphe de dépendances calculé (voir section 4.3) tiendra compte de celle-ci.

Ce calque reprendra, entre autre, les dépendances identifiées par des solutions telles que proposées par [Keller et Kar, 2001] et [Kar *et al.*, 2000], mais également les relations d'un workflow où chaque sortie d'un élément est utilisée en tant qu'entrée d'un autre (i.e. schéma Input/Output classique).

4.2.4 Calque d'abstraction ou de détails supplémentaires

Cela n'entre pas dans le cadre direct de la présente étude, mais nous notons à toute fin utile que la représentation en calques peut permettre des niveaux d'abstraction ou de détails supplémentaires. Un exemple d'abstraction serait la représentation des départements de l'entreprise et leur utilisation des différents services. A l'inverse, un calque plus détaillé pourrait représenter, par exemple, le réseau électrique.

4.3 Graphe de dépendances

Le graphe de dépendances final est l'élément central de la solution. Si les calques sont utilisés pour une représentation visuelle et faciliter l'utilisation et l'alimentation de la représentation, c'est bien ce graphe qui sera au cœur des fonctionnalités et de l'utilité de la solution. Il est construit à partir d'une analyse des relations décrites dans les différents calques et est ensuite exploité pour les analyses d'impact et de causes racines. La construction de ce graphe est décrite à la section 6.1.

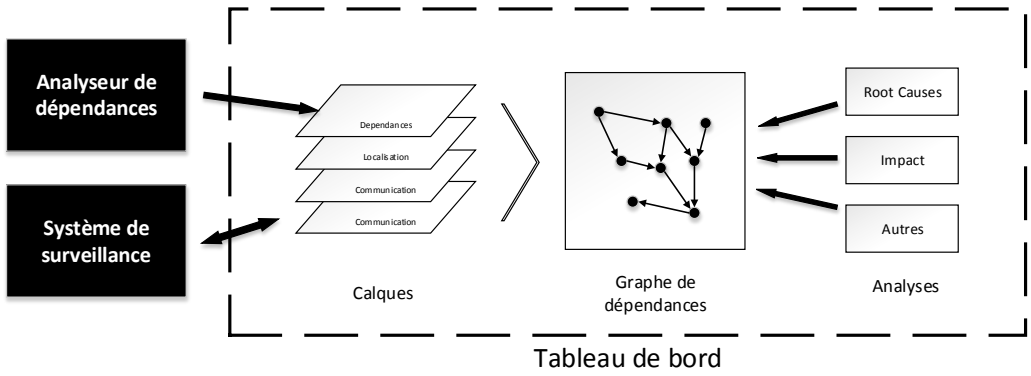

FIGURE 4.1 – Structure du tableau de bord

Chapitre 5

Représentation des éléments

La conception d'une telle solution passe par une réflexion importante : le choix de la façon de représenter les différents éléments. Il n'est pas forcément trivial de parvenir à une représentation complète et adéquate des nœuds et de leurs relations.

5.1 Représentation des nœuds

5.1.1 Représentation informatique

Nous connaissons la diversité des systèmes d'informations, aussi bien matérielle que logicielle. Tout ces systèmes, de même que leur composants, doivent être mis sur un pied d'égalité dans un graphe de dépendances puisque leur importance n'est pas définie par ce qu'ils sont mais bien par les relations qu'ils ont avec les autres éléments. La notion commune à tous est qu'ils sont des éléments inter-connectés dans un ensemble. Et c'est bien cette définition qui nous intéresse.

Ainsi, d'un point de vue informatique et algorithmique, nous ne pouvons pas les traiter différemment. Nous devons utiliser une représentation qui soit suffisamment générique que pour les représenter tous. Dans la mesure où nous manipulons essentiellement des graphes dans notre solution, nous resteront avec la simple appellation de "nœud".

Outre les informations nécessaires à la génération du graphe de dépendances, il est indispensable de pouvoir lier chaque nœud à l'élément qu'il représente afin de connaitre son état de santé en temps réel. Nous choisissons d'utiliser un identifiant unique fourni par le système de surveillance. Il nous faut également suivre son évolution, ses relations afin que la représentation soit proche de la réalité.

En ce qui concerne l'état de fonctionnement du nœud, nous devons discerner deux états distincts :

- L'état de fonctionnement actuel du nœud : c'est-à-dire l'état dans lequel il fonctionnerait s'il ne dépendait d'aucun autre élément. C'est l'état rapporté par le système de surveillance.
- L'état de fonctionnement déduit des dépendances actives : c'est-à-dire l'état du nœud étant donné l'état de ses dépendances. C'est l'état calculé par l'analyse d'impact.

L'état réel du nœud sera l'état le plus critique des deux.

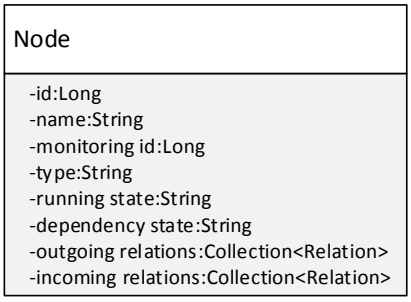

FIGURE 5.1 – Structure de donnée - Nœud

Représentation visuelle
D'un point de vue visuel et interface graphique cependant, il est intéressant voire nécessaire de pouvoir différencier chaque type de nœud suivant un code couleur et/ou une iconographie adéquate. A l'instar des relations, le détail des informations d'un nœud sélectionné peut être présenté dans un panneau latéral ou dans une infobulle afin de faciliter la lecture du graphe.

5.2 Représentation des relations

Une dépendance nait de la relation, des interactions entre deux éléments. Afin de générer un graphe de dépendances, il nous faut les cartographier et définir dans quelle mesure chacune de ces relations génère une dépendance. L'utilité de la représentation évoluera à chaque fois qu'une nouvelle relation sera identifiée.

5.2.1 Hébergement

L'une des premières relation qui nous vient à l'esprit est la relation d'hébergement. Elle représente la notion de support de fonctionnement. Tout composant logiciel, pour fonctionner, a besoin de ressources matérielles (i.e. disque, CPU, mémoire,...). Celles-ci sont généralement groupées dans un matériel (i.e. serveur hardware) que nous appellerons "hôte". Cette relation peut comporter un ou plusieurs intermédiaires logiciels (i.e. système d'exploitation), c'est notamment le cas dans les environnements de virtualisation.

Une telle relation implique une dépendance directe entre un élément et son hôte.

5.2.2 Composant/composé, Groupe, Parent/enfant

Si nous décomposons un hôte, nous y retrouvons une multitude de composants. Certains auront un impact direct sur le fonctionnement de l'hôte, d'autres n'en auront pas, mais influenceront les logiciels hébergés.

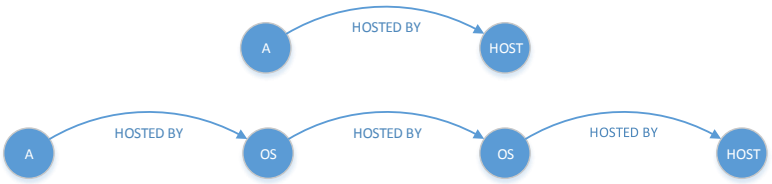

FIGURE 5.2 – Relation d'hébergement

Processeur et carte mère Le processeur et la carte mère sont de la première catégorie. Leur dysfonctionnement perturbera directement l'hôte.

Mémoire, RAID Le cas des mémoires et des RAID est légèrement différent puisqu'ils sont eux-même composés de plusieurs éléments. L'arrêt d'une barrette de mémoire dans un serveur qui en comporte plusieurs n'implique pas l'arrêt du serveur, mais le place dans un état dégradé. Par contre, l'arrêt de toutes les barrettes d'un serveur le rendra hors service. Nous ne pouvons pas dissocier chaque barrette car son impact sur l'hôte dépendra de l'état de fonctionnement des autres barrettes. Nous devons les grouper. L'état de fonctionnement du groupe dépendra, dès lors, de tous les éléments qui le composent et non plus d'un en particulier. Si au moins un des composants est arreté ou dégradé, le groupe sera dégradé. Si tous les composants sont arretés, le groupe sera hors service et donc l'hôte également. Il semble évident qu'un tel groupe ne peut comporter que des composants du même type. Nous retrouvons ce type de relation dans les clusters, la redondance ou les environnement d'équilibrage de charge (i.e. load balancing).

Disque dur, carte réseau Les disques durs auront également un impact différent suivant les données qui s'y trouvent. Prenons l'exemple suivant : un serveur contient trois disques. Sur le disque 1 est installé le système d'exploitation (SE), sur le disque 2, un service de type SGBD, et sur le disque 3, les fichiers de base de données. L'arrêt du disque 1 aura un impact sur le SE mais également sur le service, de manière transitive. A l'inverse, les disques 2 et 3 n'impacteront que le service et les données respectivement. Si nous ignorons cette relation, le graphe de dépendance ne pourra comporter la dépendance qu'elle implique, pourtant bien réelle. Cette configuration est similaire à celle de tout composant non-essentiel au fonctionnement de l'hôte, tels que les cartes réseaux, ou encore les bases de données d'un SGBD. En effet, la corruption d'une base de donnée n'entraîne pas l'arrêt du SGBD. Même si toutes les bases sont corrompues, il reste possible d'exploiter le SGBD pour toute autre tâche (e.g. restauration, création de nouvelles bases, etc.).

Nous identifions ici trois types distincts de relations :
Composant/composé : cette relation est caractérisée par :

1. l'impact direct du composant sur le composé
2. la possibilité de relation entre les composants d'un même composé
3. l'absence de relation entre le composant et le reste de l'environnement

Groupe : caractérisée par :

1. l'état de fonctionnement du groupe dépend de l'ensemble de ses éléments
2. les éléments d'un groupe sont tous du même type et peuvent avoir des relations entre eux
3. l'absence de relation entre les éléments et le reste de l'environnement

Parent/enfant :

1. l'impact direct du parent sur l'enfant
2. l'absence d'impact de l'enfant sur le parent
3. la possibilité de relation entre l'enfant et le reste de l'environnement

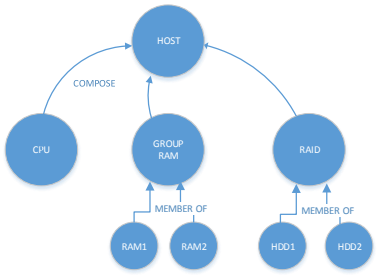

FIGURE 5.3 – Relation de composition

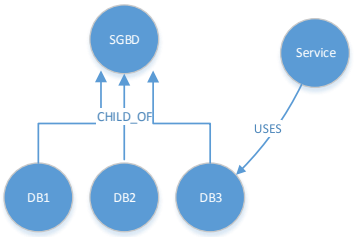

FIGURE 5.4 – Relation parent-enfant

5.2.3 Autres relations

Pour les relations décrites ci-dessus, l'impact de la dépendance induite est identifiable. En effet, l'impact direct est caractérisé par la similitude de l'état de santé du composant et du composé, du parent et de l'enfant, dans le sens de la relation. D'autres relations n'ont pas d'impact identifiable automatiquement et celui-ci doit être défini manuellement. C'est pourquoi nous utiliserons un espace d'état. Par exemple, l'impact d'une ressource sur un service qui l'utilise dépendra de la raison de cette utilisation et de la criticité de celle-ci. De même, l'impact ne sera peut-être pas immédiat (e.g. système de cache, de mémoire tampon), ou dépendra de la période de l'année durant laquelle se produit la panne. Ainsi, si certaines de ces relations peuvent être identifiées automatiquement, la quantification de leurs impacts devra se contenter d'une valeur par défaut et requerra une complétion manuelle par l'utilisateur.

5.2.4 Relations dynamiques

Les politiques de haute disponibilité pratiquées dans tous les environnements informatiques sensibles nous apportent un comportement particulier : le basculement (failover). Un élément, généralement logiciel, orchestrera une partie de l'environnement pour répondre automatiquement à la panne d'un élément. Cette réponse peut être le déplacement d'une machine virtuelle d'un espace de stockage à un autre, ou la redirection du trafic de données vers un second élément qui prendra le relais. L'objectif est bien entendu de limiter l'impact d'une panne sur l'exploitation. Notre solution se doit donc de suivre dynamiquement l'évolution de l'infrastructure. Nous apporterons une réponse à ce comportement dans la section 7.6.

5.2.5 Communication

Si la plupart des éléments d'une infrastructure ont une utilité précise dans l'exploitation de l'entreprise, d'autres se limitent à opérer la communication entre deux éléments. Nous l'avons exprimé au chapitre précédent : si un service A utilise un service B, alors A communique d'une façon ou d'une autre avec B. A dépend donc de B mais également de tous les éléments qui composent le chemin de communication entre eux. Ainsi, pour être complète, chaque relation doit comporter une information indiquant le type de communication utilisé. Celui-ci peut très bien être la relation elle-même [1] (e.g. hébergement, composant/composé, etc.) ou un autre type de relation. Nous introduisons donc des relations de type "communication" qui représenteront, entre autre, le réseau TCP-IP [2] ou encore des flux de données. Ces relations peuvent être bidirectionnelles.

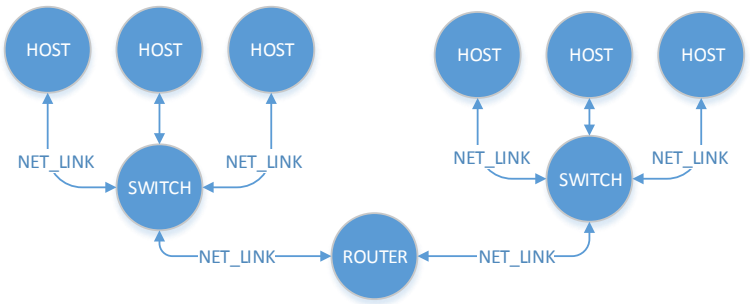

FIGURE 5.5 – Relation de communication TCP/IP

Notons que, malgré la présence d'un lien TCP/IP entre deux éléments, ceux-ci peuvent ne pas pouvoir communiquer en raison d'une configuration particulière (e.g. règles d'accès d'un firewall, redirection de port, etc.), qu'elle soit volontaire ou non. Il est indispensable que les liens présents dans le graphe représentent les communications fonctionnelles. Il serait par exemple nécessaire de scinder un nœud de type "switch" en sous-groupes correspondant chacun à un VLAN. Dans un souci de délai, nous décidons de reporter l'analyse de cette problématique dans une étude ultérieure dédiée à l'analyse des dépendances liées aux configurations.

5.2.6 Représentation informatique

Dans la structure de donnée qui représentera une relation (fig. 5.6), nous avons besoin de plusieurs informations pour générer le graphe de dépendance.

- La relation implique-t-elle une dépendance ?
- Quelle est la direction de la dépendance induite ?
- Quel est le chemin de communication exploité par la relation ?
- Quel est le type de la relation ?
- Quel est l'espace d'état ?

1. Ceci dans le simple objectif de ne pas alourdir la représentation par une granularité inutile
2. A noter qu'un réseau d'antennes WI-FI pourra être représenté par un ou plusieurs groupes

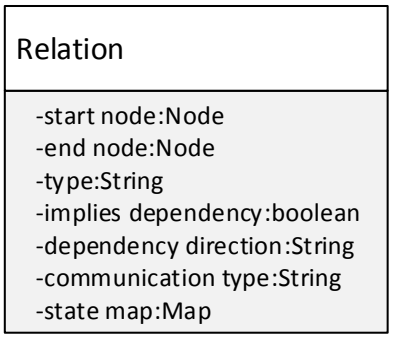

FIGURE 5.6 – Structure de donnée - Relation

Représentation visuelle
D'un point de vue visuel, toute ces relations sont des arcs dans un graphe et ne comportent donc pas de particularité. Dans un souci de faciliter la lecture, nous pourrions établir une gamme de couleur pour chaque type de relation. Le détail des données d'une relation sélectionnée peut être affiché dans un panneau latéral ou une infobulle.

5.3 Représentation d'une panne

La représentation d'une panne est essentiellement visuelle et vise uniquement à alerter l'administrateur. L'un des principaux objectifs de cette représentation doit être de réduire au minimum le délai d'analyse et d'intervention (voir section 1.2). Pour atteindre cet objectif, la représentation se doit de :

- Ignorer les éléments sans aucun rapport avec la panne concernée
- Présenter toutes les causes possibles (analyse de causes racines)
- Présenter l'impact de la panne afin de faciliter la prise de décision

L'espace d'état que nous choisissons ainsi que la coloration est relativement classique :

Etat	**Couleur**
Unknown	gris
Ok	vert
Warning	jaune
Degraded	orange
Critical	rouge
Down	noir/rouge clignotant

TABLE 5.1 – Espace d'état de fonctionnement des nœuds

Pour faciliter le travail d'investigation, il est nécessaire de présenter les deux états de fonctionnement du nœud. Ainsi, la couleur du nœud représente l'état de fonctionnement actuel, tandis que la couleur du contour du nœud représente l'état calculé à partir des dépendances.

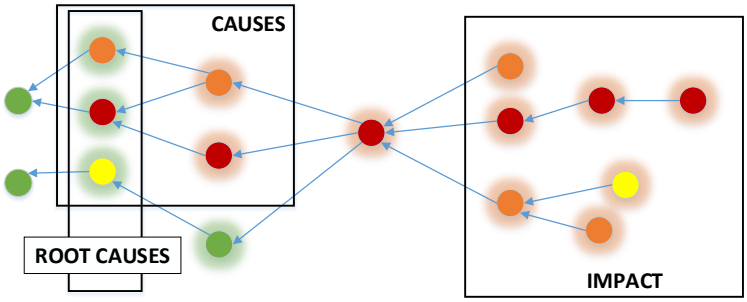

FIGURE 5.7 – Représentation d'une panne

Chapitre 6

Analyses des éléments

Une fois la représentation conçue, nous pouvons l'exploiter pour générer le graphe de dépendances.

6.1 Génération du graphe de dépendance

6.1.1 Dépendances induites

Comme nous l'expliquions dans la section 5.2, nous identifions plusieurs types de relation qui induisent une dépendance :

- HOSTED BY implique une dépendance directe entre un élément et son hôte
- COMPOSED OF implique une dépendance directe entre un élément et son composant
- MEMBER OF implique une dépendance indirecte entre un groupe et les éléments qui le composent. L'impact dépendra de l'état de tous les éléments
- CHILD OF implique une dépendance directe entre un enfant et son parent, ainsi qu'une dépendance vers tous les éléments présents sur le chemin de communication
- USES implique une dépendance entre un élément et le service, la ressource qu'il utilise, ainsi qu'une dépendance vers tous les éléments présents sur le chemin de communication

Etant donné la structure de donnée utilisée pour les relations (fig. 5.6), nous savons si celles-ci impliquent une dépendance ou non, quelle sera la direction de celle-ci et si elles utilisent un chemin de communication différent. Il ne nous est donc pas nécessaire de différencier chaque relation qui induit une dépendance. Cette distinction sera par contre utilisée dans les analyses d'impact et de causes racines (Sect. 6.2 et 6.3).

6.1.2 Gestion des circuits

Nous l'avons vu, la présence de circuits dans un graphe de dépendances dénoncent un défaut de conception dans l'infrastructure. Il est couteux de vérifier la création de circuit à chaque ajout d'une nouvelle dépendance dans la mesure où nous ne connaissons pas à l'avance l'ordre dans lequel les relations seront traitées, ni leur direction.

La théorie des graphes nous proposent plusieurs algorithmes pour la détection de circuits qui varient en complexité. Dans tous les cas, il est moins couteux d'exécuter une fois un algorithme qui liste l'ensemble des circuits que n fois celui-ci ou un autre (n étant le nombre de dépendances). Le nombre d'itération du processus de correction de ces circuits sera également grandement réduit.

Notre solution vérifiera la présence de circuit après la génération du graphe (voir section 6.6).

6.1.3 Algorithme de génération

Afin d'alimenter le graphe de dépendances, il nous faut parcourir toutes les relations décrites dans la représentation et en déduire les dépendances sur base de ce que nous avons identifié.

Il sera donné à l'utilisateur de créer de nouveaux types de relations à condition de renseigner si celles-ci induisent une dépendance ou non.

L'expérience acquise par l'exercice de la fonction d'administrateur nous permettra de compléter les types de relations reconnus nativement par la solution.

Une caractéristique importante à prendre en considération est la possibilité d'un élément de communiquer au travers d'un autre. Notamment, les services communiquent sur le réseau TCP/IP au travers de leur hôte. Le graphe de communication ne comprend pas de lien "linked to" entre le service et son hôte, il ne sera donc pas possible de trouver un chemin n'exploitant que ce type de relation. Si la relation entre deux éléments permet une communication à travers l'hôte, le chemin à retrouver est différent.

L'algorithme utilisé sera le suivant :

Require: completeGraph : a graph containing all nodes and relations
Ensure: a new graph containing all nodes and the dependencies is created

```
1:  function GenerateDepGraph(completeGraph)
2:      for all relation in completeGraph.allRelations do
3:          if relation implies dependency then
4:              if relation.dependecyDirection is reverse then
5:                  startNode ← relation.to
6:                  endNode ← relation.from
7:              else
8:                  startNode ← relation.from
9:                  endNode ← relation.to
10:             end if
11:             dependency ← createDependency(startNode,endNode)
12:             newGraph.add(dependency)
13:             if relation.communicationType != relation.type then
14:                 if relation.communicateThroughHost is true then
15:                     nodes ← completeGraph.getNodesInPath(startNode.host,
    endNode.host, relation.communicationType)
```

```
16:                 else
17:                     nodes ← completeGraph.getNodesInPath(startNode,
    endNode, relation.communicationType)
18:                 end if
19:                 for all node in nodes do
20:                     dependency ← createDependency(startNode,node)
21:                     newGraph.add(dependency)
22:                 end for
23:             end if
24:         end if
25:     end for
26: return newGraph
27: end function
```

Nous vérifions que le chemin de communication est différent du type de la relation (10) puisque dans le cas contraire, le chemin sera forcément de longueur 1 et comportera uniquement les nœuds $relation.from$ et $relation.to$, soit la relation elle-même, qui a déjà généré une dépendance à la ligne 11.

Une fois le graphe de dépendance généré, nous pouvons pratiquer les analyses qui apportent son utilité au tableau de bord.

6.2 Analyse d'impact

Nous avons défini deux états de fonctionnement distincts pour les nœuds du graphe. L'un représentant l'état de fonctionnement actuel, tel que rapporté par le système de surveillance, l'autre représentant l'état déduit des états de chaque dépendance. Dans un souci de clarté, nous appellerons "état de fonctionnement" le premier, et "état de dépendance" le second.

Une analyse d'impact sera démarrée lorsqu'un changement d'état de fonctionnement sera détecté par le système de surveillance pour un élément et mettra à jour les états de dépendance des nœuds qui en dépendent suivant les types de relations qu'ils ont.

Ce sera l'état de fonctionnement d'un élément qui générera une analyse mais c'est bien l'état de dépendance qui sera modifié par l'analyse. En effet, le premier ne pourra être modifié par une quelconque analyse puisqu'il représente l'état actuel de l'élément.

Une fois les états de dépendance mis à jour, présenter l'impact du changement d'état initial reviendra à lister les nœuds de l'arbre de dépendance dont le premier élément est la racine.

Notons qu'un second type d'analyse d'impact peut naître d'autres évènements, tels que l'ajout ou la suppression d'un élément ou d'une relation. Nous discuterons ce point dans la section 6.4.1 qui traite des simulations d'évènements.

Pour que le tableau de bord soit efficace, l'état de dépendance devra toujours représenter l'état le plus critique compte tenu des états de chaque dépendance. Dans le cas où plusieurs éléments influent simultanément sur un nœud, et que l'un des éléments revient à l'état "Ok", il sera couteux de déterminer s'il faut mettre à jour ou non l'état de dépendance. Pour pallier à

ce coût, nous n'utiliserons pas un seul état de dépendance mais un ensemble d'état dans lequel l'identifiant de la dépendance est associé à l'état qu'elle génère.

Pour afficher ou identifier l'état de dépendance d'un élément il nous faudra connaître si la relation implique un calcul d'impact particulier. Si nous reprenons les types de relations que nous avons identifiés au chapitre 5, seule la notion de groupe implique un traitement différent. A savoir, la nécessité de consulter l'état de tous les membres du groupe pour en déterminer l'état de dépendance. Ainsi, pour tous les éléments à l'exception des groupes, il faudra retrouver l'état le plus critique parmi l'ensemble. Pour les groupes, il faudra vérifier tous les états de dépendance des éléments de l'ensemble. Si au moins l'un d'entre eux est "degraded" ou supérieur, l'état du groupe sera "degraded", si seule une minorité d'entre eux est "ok", le groupe sera "critical". Enfin, si aucun d'entre eux n'est "ok", le groupe pourra généralement être considéré comme "down".

6.2.1 Algorithme

L'algorithme d'analyse d'impact est fondamentalement un algorithme de parcours de graphe orienté non-pondéré et sans circuit classique. L'absence de circuit et la nécessite de parcourir l'entièreté du graphe afin de diffuser l'impact à tous les nœuds dépendants, oriente le choix de l'algorithme de parcours sur BFS. En effet, prenons l'exemple suivant :

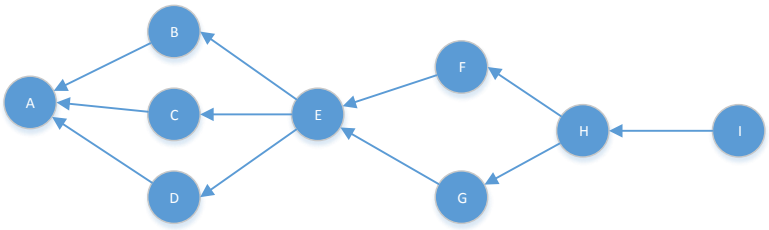

FIGURE 6.1 – Analyse d'impact - BFS vs DFS

Dans le cadre d'une analyse d'impact, nous ne pouvons pas nous arrêter à un nœud simplement parce qu'il a déjà été visité par l'algorithme. Le nouvel impact propagé peut être plus critique que le précédent. Avec un parcours DFS au départ du nœud A, les nœuds E, F, G seront parcourus trois fois, et les nœuds H, I cinq fois. Avec un parcours BFS, seuls les nœuds E, H seront atteints respectivement trois et deux fois. L'application d'un parcours DFS adapté à ce cas nécessitera systématiquement la comparaison de l'état de dépendance du dépendant au nouvel état.

Durant le parcours, il nous suffira de mettre à jour l'état de dépendance correspondant à l'arc parcourus parmis l'ensemble d'état de dépendance du nœud atteint. Nous itérons d'abord sur les premiers dépendants car pour le premier nœud, c'est bien l'état de fonctionnement qu'il faut propager et non l'état de dépendance.

```
Require: startNode : the node from which to start the analysis
Ensure: the dependency state of all child nodes is updated
  function ImpactAnalysis(startNode)
      queue ← create new queue
      for all dependant in startNode.getDependants() do
          dependant.addDepState(parentNode.id,startNode.runningState)
          add dependant to queue
      end for
      while queue is not empty do
          node ← queue.dequeue()
          for all dependant in node.getDependants() do
              dependant.addDepState(node.id,node.depState)
              add dependant to queue
          end for
      end while
  end function
```

La fonction *addDepState* écrasera l'état de dépendance présent correspondant à l'identifiant du nœud *parentNode* par un nouvel état en se basant sur l'espace d'état contenu dans la relation de dépendance et l'état *parentNode.depState* passé en argument.

6.3 Analyse de causes racines

L'objectif de l'analyse de causes racines est d'identifier quels nœuds sont à la source de la panne analysée. C'est-à-dire qu'il faut parcourir toutes les chaînes de dépendances au départ de l'élément concerné. Chacune de ces chaînes sera considérée comme une chaîne causale si au moins l'un des nœuds de celle-ci est dans un état différent de *ok*. La cause racine de chaque chaîne causale sera le dernier nœud accessible dont l'état de fonctionnement n'est pas *ok* et l'état de dépendance est *ok*.

Notons qu'un nœud de type "groupe" n'aura pas d'état de fonctionnement puisqu'il représente une notion abstraite. Dans ce cas, nous utiliserons l'état de dépendance.

Rappelons que l'état de fonctionnement défini par le système de surveillance est établi sur base des tests opérés par celui-ci. Si ces tests ne couvrent pas exhaustivement l'étendue des défaillances d'un élément, une panne de ce dernier ne sera pas forcément détectée. A l'inverse, l'état de fonctionnement d'un élément est probablement déjà lié à la défaillance d'une de ses dépendances et il ne sera pas toujours possible, pour le système de surveillance, de vérifier l'état de fonctionnement de tous les éléments (e.g. il n'est plus possible de vérifier l'état du processeur lorsque le serveur ne répond plus). Ce sont là les limites des systèmes de surveillance actuels. Il est, dès lors, important de garder à l'esprit que la liste des causes racines établie par l'analyse est à mettre en relation avec l'historique des tests de surveillance des éléments incriminés, ainsi que de leurs propres dépendances.

Notons également que notre solution permet de prendre en considération des éléments qui ne sont pas présents dans le système de surveillance et donc, dont l'état de fonctionnement n'est modifié que manuellement. Si cette modification n'a pas encore été opérée, cela ne signifie

pas que l'élément fonctionne correctement, ni qu'il est toujours défaillant. Il sera donc utile d'alimenter un ensemble de "causes racines plausibles" de ceux-ci afin qu'il ne soit pas oubliés lors d'une investigation de la panne. Nous ne pouvons pas les comptabiliser avec certitude dans une analyse de causes racines. Leur utilité est principalement liée à l'analyse d'impact.

6.3.1 Algorithme

En reprenant l'exemple de la figure 6.1, nous voyons que cela a du sens de s'interroger sur le choix de l'algorithme de parcours. Il sera inutile de parcourir une seconde fois un nœud dont nous avons déjà découvert toutes les causes racines. De plus, il ne s'agira pas de parcourir tout le graphe, mais uniquement de trouver les nœuds qui répondent à la condition $node.depState = ok$ and $node.runningState! = ok$, ce qui représente notre cas de base. Nous opterons pour DFS qui nous permettra de directement chercher les causes racines de chaque nœud atteint. Enfin, il est inutile de parcourir les dépendances d'un nœud dont l'état de dépendance est *ok* pour des raisons évidentes.

Require: startNode : the node from which to start the analysis
Ensure: a list of all the identified root causes and a list of possibles root causes are filled

```
function StartRootCausesAnalysis(startNode)
    RClist ← create new list
    possibleRClist ← create new list
    visitedNodes ← create new list
    for all dependency in startNode.getDependencies() do
        RootCausesAnalysis(dependency)
    end for
end function
function RootCausesAnalysis(node)
    if node not in visitedNodes then
        add node to visitedNodes
        if node.manuallyMonitored is true then
            add node to possibleRClist
        end if
        if node.depState == ok then
            if node.runningState != ok and node.manuallyMonitored is false
and node.type != "group"  then
                add node to RClist
            end if
        else
            for all dependency in node.getDependencies() do
                RootCausesAnalysis(dependency)
            end for
        end if
    end if
end function
```

6.4 Requêtes sur le graphe, simulations et échanges de données

6.4.1 Simulations

L'une des grandes utilités des modélisations numériques est la possibilité de pouvoir altérer le modèle à souhait, à frais réduits et sans que cela n'impacte l'original. Nous ne pouvons ignorer ce potentiel dans des contextes d'utilisation critiques.

Cependant, nous ne pouvons modifier la modélisation elle-même ni son graphe de dépendance puisque ceux-ci présente également l'état de santé du système en temps réel. Nous créerons donc, pour chaque nouvelle simulation, une copie de la représentation dont la durée de vie sera équivalente à celle de la simulation.

L'intérêt des simulations dans notre solution est multiple. D'une part, nous pouvons évaluer l'impact de la variation de l'état de santé d'un ou plusieurs éléments. Cette évaluation pourra servir à planifier une intervention ou afin de prioritiser l'allocation de budget à l'amélioration des éléments concernés.

D'autre part, nous imaginons aisément une simulation où l'on évalue plusieurs transformations de l'infrastructure en vue de répondre à un nouveau besoin, dans le but de consolider l'existant, ou encore d'apporter une correction à une faute de design.

Nous y voyons également un intérêt pédagogique. Ces simulations pourraient servir de base pour l'enseignement ou la formation. Elles facilitent la démonstration de cas théoriques et permettent à des apprentis d'expérimenter des transformations, ou des conceptions.

Ces simulations exploiteront les mêmes concepts et fonctionnalités déjà évoqués dans les précédents chapitres.

6.4.2 Système de requête

Nous ne pouvons prévoir les besoins de tous les utilisateurs. En effet, si nous prévoyons de présenter un certain volume d'information (e.g. graphe d'impact, de dépendance, etc.), nous savons qu'une exploitation informatique génère un besoin croissant en données statistiques ou de calculs permettant de répondre à des questions toujours plus complexes.

Une méthode pour répondre à ce besoin est de prévoir un système et un langage de requête laissant libre cours à l'imagination de l'utilisateur. Nous avons à notre disposition une gamme de langages de ce type et il n'est pas utile à ce stade d'en inventer un nouveau. Nous pouvons même implémenter plusieurs de ces langages.

SQL est très répandu. Cypher peut, lui, compléter l'attirail de fonctionnalités par l'exécution, à la demande, de recherche de chemins complexes comportant plusieurs types de relations et de nœuds, de fonctions statistiques, de mise à jour de données en masse ou même d'ajout de nouveaux champs de données pour un nœud donné.

Prenons quelques requêtes d'exemple :

- `MATCH (n)-[:DEPENDS_ON]->(m) WHERE id(n)=1 RETURN m`

  Retourne la liste des nœuds dont le nœud d'id 1 dépend (relation de type "DEPENDS_ON").

- `MATCH (n)-[r]->(m) WHERE id(n)=1 RETURN r`

  Retourne la liste des relations qui partent du nœud d'id 1.

- `MATCH (n)-[r]->(m) WHERE id(n)=1 RETURN count(r)`

  Retourne le degré extérieur du nœud d'id 1.

- `START from=node(1), to=node(15) MATCH path=(from)-[rels*]->(to) WHERE all(r in rels WHERE type(r)=NETWORK) RETURN path"`

  Retourne l'ensemble des chemins du nœud 1 au nœud 15 ne contenant que des relations de type "NETWORK"

Un autre principe simple de système de recherche est l'inclusion d'un champ de saisie de texte dans l'interface permettant d'y rechercher des nœuds et des relations par leur identifiant, leur nom ou toute autre donnée qu'ils contiennent. Le résultat pourrait être mis en évidence dans la visualisation ou présentés dans une nouvelle fenêtre.

Enfin, nous pouvons mettre en place plusieurs filtres prédéfinis qui permettrait d'afficher ou non un ensemble ou un type particulier de nœuds ou de relation.

6.4.3 Import/Export de données et interfaces

S'il est utile de manipuler la solution telle qu'elle est présentée, nous souhaitons, de manière contrôlée, ouvrir l'accès aux données afin que l'utilisateur puisse les exploiter comme bon lui semble et ce, au delà des fonctionnalités prévues par la solution.

6.5 Problématique de couvrir l'étendue des dysfonctionnements possibles

Outre le fait de présenter un état de santé, il faut l'analyser et voir ce que cela implique ou pourrait impliquer sur le reste du système. L'infrastructure gérée par le tableau de bord évoluera suivant les pannes. Un système qui comporte deux serveurs de mail n'est pas le même que celui qui n'en a qu'un. Lorsqu'un des deux sera en panne, le système ne sera plus le même. Si la nature d'un des éléments du système change (panne, etc), le système entier change.

L'impact de la défaillance d'un élément sur le système ne dépend pas que de ses relations. Celui-ci pourrait être décupler s'il survient simultanément avec la défaillance d'un ou plusieurs autres éléments.

Chaque composant matériel ou logiciel, présent dans une infrastructure, apporte avec lui son lot de dysfonctionnements. Il est dès lors impossible de couvrir l'ensemble des pannes

possibles. L'utilisation de relations de dépendances permet d'abstraire un grand nombre de pannes.

Le domaine de l'intelligence artificielle pourrait nous apporter des pistes de solution pour cette problématique. Une autre idée serait d'alimenter une base de connaissance nous permettant d'identifier des comportements récurrents et ainsi de prédire les conséquences possibles.

Dans tous les cas, cette problématique requiert d'avoir une connaissance et une conscience approfondie de l'infrastructure concernée et, éventuellement, de l'historique d'état et de performance des éléments qui la composent.

6.6 Mise en évidence des sections critiques

A l'image de la problématique décrite ci-dessus, il nous semble impensable, dans cette première version de solution, de pouvoir confronter l'infrastructure à une liste exhaustive de défaut de conception ou de bonnes pratiques.

Nous avons cependant connaissance de certains modèles et certaines caractéristiques qui s'apparentent à des erreurs.

La première, déjà citée à plusieurs reprises, est la présence de circuits dans le graphe de dépendances. Ainsi, il sera nécessaire de prévoir cette vérification après chaque mise à jour du graphe. Dans cette première version, nous nous contenterons d'avertir l'utilisateur.

Une seconde caractéristique est évidement la présence d'un nombre élevé de dépendants pour un élément donné, si celui-ci n'est pas un groupe (i.e. cluster, redondance,etc.). Ceci est aisément calculable. Cet élément mériterait une attention particulière.

Enfin, en ajoutant à chaque nœud une information de niveau d'alerte ou de criticité pour l'entreprise, nous pourrions opérer un ensemble de vérifications sur, par exemple, tous les éléments de la chaine de dépendances d'un nœud dont la criticité est élevée.

Chapitre 7

Intégration dans un système distribué

Notre représentation à présent définie, il nous faut l'alimenter avec les données de l'entreprise.

La tâche d'alimentation d'une telle modélisation est incommensurable [Ensel, 1999]. Cependant, il existe énormément de méthodes pour nous permettre d'aider l'utilisateur dans cette tâche d'identification de la salle machine, les éléments qui la composent et leurs relations [Brown *et al.*, 2001] [Kar *et al.*, 2000] [Keller et Kar, 2001].

Pour ce faire, nous avons mentionné plusieurs méthodes dans les chapitres 1 et 3. Notamment le fruit des recherches de Kar et Keller. Le système de surveillance est également une source d'information indispensable. Ensuite, pour les informations qui n'ont pu être automatiquement détectées, il sera bon d'implémenter des fonctionnalités d'interface graphique et d'échange de données.

7.1 Problématique du niveau de détail [USNRC, 1998, ChI-2]

Lorsqu'il est question de représenter un système existant, une notion importante est celle des limites qui définissent ce système. Il est indispensable de se demander ce qu'il sera nécessaire d'inclure dans cette représentation et ce que l'on pourra abstraire ou ignorer.

7.1.1 Limite externe

Nous définissons limite externe celle qui détermine ce qui fait partie du système et ce qui n'en fait pas partie. Cette limite peut avoir plusieurs natures : géographique, politique, technique, etc.

Dans une infrastructure hébergée entièrement en interne, la limite externe est aisément établie et est généralement la ligne internet du fournisseur d'accès. Cependant, à l'heure où l'utilisation du Cloud est de plus en plus présente, que devient cette limite ? Inclut-elle le réseau

de télécommunication qui relie l'utilisateur du service hébergé ? Faut-il également y inclure l'infrastructure d'hébergement ?

L'établissement de cette limite va dépendre de ce que l'on souhaite surveiller, de quels aspects de performance seront pertinent dans le contexte d'utilisation.

7.1.2 Limite interne

La limite interne, elle, va définir le niveau de détail du système que l'on souhaite. Fondamentalement, un système est composé de sous-systèmes, eux-même composés de sous-sous-systèmes.

Ces sous-systèmes interagissent entre eux, généralement de manière très complexe. Ainsi, un système n'est pas seulement égal à la somme de ses composants , il est également défini par leurs interactions.

A quel point faudra-t-il détailler la représentation ? Cela a-t-il un sens d'y inclure les composants électroniques de la carte mère, ou les secteurs d'un disque dur ?

Ce sera le cas pour un système qui exploite des micro-environnements mais pas pour d'autres.

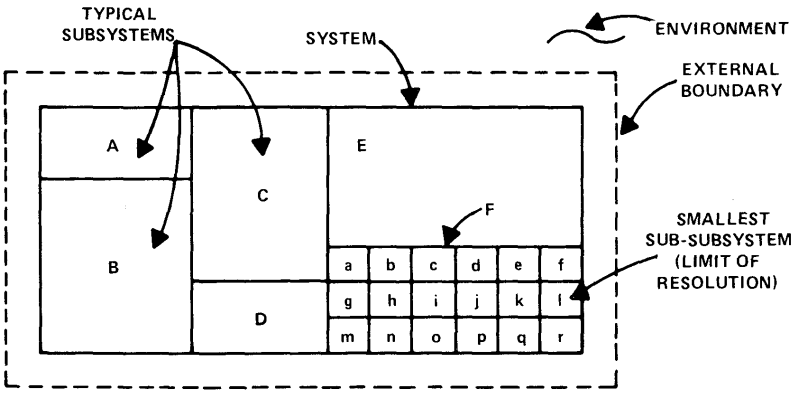

FIGURE 7.1 – System Definition : External and Internal Boundaries [USNRC, 1998, Fig. 1-4]

Lors du développement de notre solution, nous avons gardé à l'esprit la disparité des systèmes d'informations actuels et les diverses utilisations de ceux-ci. Nous avons, dès lors, abstrait toute notion de types d'éléments dans le fonctionnement fondamental de la solution en nous concentrant sur ce qui importe le plus : les relations. Le devoir de réponse à ces questions reviendra donc à l'utilisateur suivant les besoins rencontrés.

7.2 Identification des serveurs et du réseaux

Il est peu probable que notre solution s'avère nécessaire dans une infrastructure qui ne contient pas déjà un système de surveillance. C'est aussi ce dernier qui nous renseignera sur l'état de santé des éléments.

Nous prévoyons donc un dialogue régulier entre le système et le tableau de bord. Il n'est pas ici question de modifier la configuration ou les données du système mais d'y accéder en lecture pour y récupérer un maximum d'information. Ces systèmes proposent généralement des interfaces (e.g. API, REST, etc.) qui permettent de récupérer les données dans des formats d'échange standardisés tels que XML ou JSON.

Nous prendrons l'exemple de Nagios car il présente une particularité : un hôte est un élément de l'infrastructure et un service est un test effectué sur cet hôte. Nagios comprend également la notion d'hôte parent qui détermine l'accessibilité d'un hôte au travers de ses ascendants. Il nous est ainsi possible de déterminer les liens réseaux, de même que les notions d'hébergement. En effet, un élément parent de type "switch" ou "router" dénote un lien réseau, tandis qu'un parent de type "VMWare Server" permet d'identifier une virtualisation et donc une relation "HOSTED BY". Un hôte peut également être associé à plusieurs parents et ainsi potentiellement annoncer la présence de redondance suivant les types des éléments parents.

Nous pouvons également exploiter les données de solutions de gestion de parc informatique et d'inventaire qui peuvent avoir déjà fait l'effort de sonder l'infrastructure à la recherche de matériels informatiques.

7.3 Identification des services et applications

L'identification des services peut s'avérer plus complexe. En effet, il n'est pas rare que les systèmes de surveillance ignorent la distinction entre matériel, système d'exploitation et service. Un élément peut être identifié sous le type "serveur" indifféremment de ce qu'il est réellement. Cette distinction apporte souvent peu d'intérêt en terme de surveillance.

Un moyen de pallier à ce manque de distinction est de s'intéresser aux tests de vérification eux-même. Si nous y trouvons des tests d'utilisation de ressource processeur et que l'hôte n'est pas lié à un parent de type "server", nous pouvons supposer que nous sommes en présence d'un serveur matériel. Si nous trouvons des tests d'état de file de messages, nous avons probablement un service logiciel. Un hôte peut, bien évidemment regrouper les deux et il sera alors nécessaire de le scinder en deux nœuds dans la représentation.

Les gestionnaires d'inventaire, au même titre que les solutions de suivi de problème seront ici aussi très utiles pour rassembler un maximum d'information. Outre la liste des éléments et leur hôte nous pourrons y retrouver des informations détaillées telles que les versions, les licences, les SLAs et même établir des profils de causes de panne pour les composants qui génèrent un nombre important de problèmes. Il faudra cependant établir des filtres afin de ne pas polluer la représentation avec nombre de logiciels sans importance (e.g. media player, décompresseur et explorateur de fichiers, etc.).

Nous attirons l'attention sur le fait que, pour que la représentation soit la plus exacte possible, une finesse de détail est indispensable au niveau de l'analyse des données du système. Il n'est pas utile pour la compréhension du présent chapitre, de présenter ici l'ensemble de ces détails, mais notons que des erreurs d'identification à cette étape génèreront des incohérences dans le graphe de dépendances final (e.g. présence de fausses dépendances ou absence de dépendances réelles).

7.4 Identification des dépendances

Nous pouvons d'ores et déjà identifier des dépendances sur base de l'analyse des relations. Pour la majeure partie des autres, nous avons des sources d'informations que nous pouvons sonder à l'aide d'agents logiciels.

D'une part, nous trouverons ces informations dans l'analyse des prérequis d'installation des logiciels. Ces derniers ne renseignent pas la localisation de ces prérequis, et ne nous permettent donc pas d'établir directement une relation. Il nous indiquent qu'une dépendance existe et qu'il faudra l'analyser afin de la définir précisément.

D'autre part, nous pouvons récupérer de nombreuses relations dans la configuration des services. Ceux qui utilisent une base de données, comportent une configuration de type "data source" renseignant précisément le type, l'adresse et la sécurité du service exploité ainsi que les bases de données concernées. Il en est de même pour l'utilisation de services de messagerie, d'impression, d'annuaire, de résolution de nom et bien d'autres.

Ces configurations ne sont pas toujours accessibles. Soit par leur emplacement peu orthodoxe, soit par leur sécurité ou simplement par manque d'accès distant. Dans ce cas, il est aussi possible de sonder les échanges réseaux entre les éléments. Les relations entre éléments étant la principale préoccupation de cette étude, ces échanges sont une mine d'informations. Ils nous permettent d'établir les calques de communication, mais aussi de nombreuses dépendances qui n'auraient pas été retrouvées par l'analyse des configurations

7.5 Architecture

Toujours dans l'idée de diversité des systèmes d'information, nous prévoyons une architecture qui permet de s'interfacer avec le système de surveillance au moyen de plugins. Si nous pouvons développer un nombre croissant d'interfaces dédiées à différentes solutions de surveillance, nous pouvons également autoriser l'utilisateur à créer sa propre interface pour une gestion plus fine des échanges de données, ou si le système est lui-même un produit interne. Ce même système d'interface peut aussi permettre d'intégrer les données d'une solution d'identification de dépendance potentiellement déjà présente.

De même, nous souhaitons ouvrir les possibilités de création d'interfaces graphiques en exposant des API permettant de gérer les données et l'exécution de la solution.

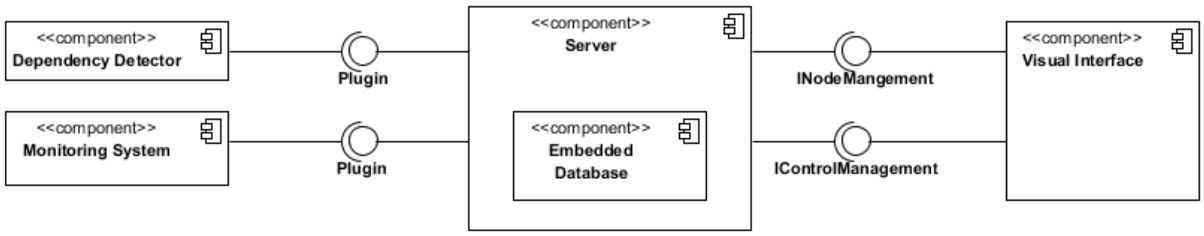

FIGURE 7.2 – Architecture de la solution

7.6 Maintenance du graphe

La maintenance de la représentation est une tâche tout aussi importante que la première génération. Suivant une fréquence adaptée et paramétrée par l'utilisateur, il sera question de comparer une nouvelle identification à la représentation. Nous souhaiterions que la représentation s'adapte d'elle-même à l'évolution de l'infrastructure. Cela présente quelques difficultés. La comparaison peut rapporter plusieurs types de différence :

- Une donnée majeure de l'élément est modifiée : nous associons un nœud à son correspondant réel sur base de plusieurs données généralement fiables (e.g. identifiant dans le système de surveillance, numéro de série). Si au moins l'une d'entre elles ne correspond plus lors de la vérification, nous pouvons supposer l'élément déplacé ou supprimé. Dans les deux cas, le plus simple reste de supprimer le nœud de la représentation et de le recréer lorsque l'élément est à nouveau détecté. Nous devons faire attention à ne pas associer l'inaccessibilité d'un élément à un changement de donnée. Dans ce dernier cas, le nœud correspondant sera placé en état "down".
- Les relations d'un élément sont modifiées : Ceci peut être la conséquence du déclenchement d'un système de haute disponibilité dont l'action corrective aura par exemple été de déplacer une machine virtuelle vers un nouvel hôte. Suivant le type de relation, l'ajustement sera différent. Une telle modification aura probablement un impact sur les dépendances et requerra une adaptation.
- Une donnée liée à l'accessibilité de l'élément est modifiée : l'adresse IP ou le nom d'hôte sont de bons exemples. La modification de l'un d'entre eux peut rendre l'élément inaccessible pour ses dépendants (suivant le type de communication de la dépendance). Il pourra alors être utile d'alerter la nécessité de vérifier les configurations de ces dépendants ainsi que l'accessibilité de l'élément par tout ceux qui lui sont liés par une relation.
- Une donnée mineure de l'élément est modifiée : ces données mineures sont de simples informations qu'il est utile de présenter (e.g. SLAs). Il faudra ici simplement mettre à jour la donnée dans le nœud.

Si un conflit de données ne peut être résolu par notre solution, il sera nécessaire de faire appel à l'utilisateur afin d'assurer que les données soient exactes. De plus, si au moins une modification importante est détectée, et n'est pas liée à une panne (i.e. les nœuds sont dans un état fonctionnel), il sera nécessaire de re-générer la portion du graphe de dépendances concerné par la différence.

7.7 Elements d'interface

La présence d'un système de surveillance ne nous est pas garantie lorsque la solution sera intégrée dans une infrastructure. Pour ces cas, mais également si l'objectif d'utilisation est de définir le design d'une installation inexistante afin de l'analyser, nous devons intégrer des facilités dans les interfaces d'interaction à disposition de l'utilisateur.

7.7.1 Définition de patrons de représentation

Nous connaissons plusieurs composants qui se retrouvent généralement dans une infrastructure. Pour ceux-là, nous pouvons définir un ou plusieurs patrons de représentation, généralement appelés "*template*". Par exemple :

– Domain controller, mail server, système de stockage

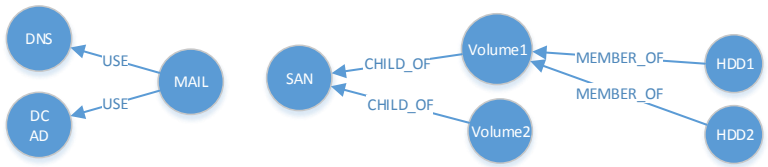

FIGURE 7.3 – Template - Mail, Domain Controller et SAN

– Virtualisation

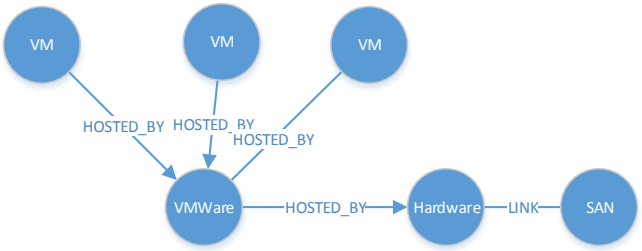

FIGURE 7.4 – Template - Virtualisation

– Haute disponibilité

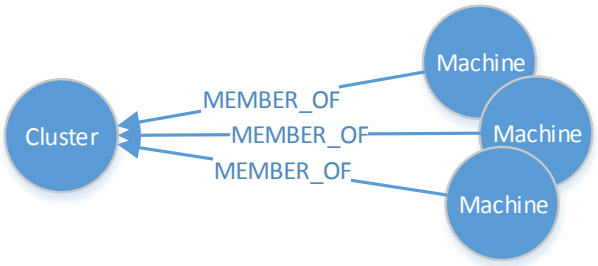

FIGURE 7.5 – Template - Cluster

– Réseau TCP/IP

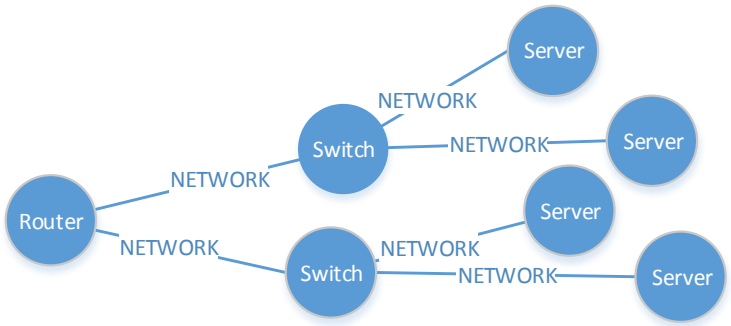

FIGURE 7.6 – Template - Réseau

7.7.2 Assistant de création

Une aide interactive répandue est l'assistant de création. Celui-ci permet de guider l'utilisateur dans le processus de création d'un élément par un découpage en étapes de ce processus et accompagné de formulaires simples. Nous pouvons ainsi nous assurer que les données encodées sont complètes et correctement formatées. Ces assistants sont déterminés par le type d'élément qu'ils doivent créer.

Nous pouvons bien entendu pré-remplir les formulaires de création afin de faciliter la tâche. Ce pré-remplissage peut se baser sur le contexte d'utilisation, nous pouvons associer le déclenchement d'un assistant à une interaction graphique précise. Par exemple :

- L'action de tracer un trait entre deux nœuds détermine l'élément de départ et de fin d'une relation et donc sa direction.
- Si la visualisation est actuellement filtrée pour n'afficher qu'un type de relation, celle-ci peut également être pré-définie.
- L'assistant de création d'un élément peut permettre de directement créer les relations correspondantes et vice-versa.

7.7.3 Import de données par fichier

Lorsque l'on parle d'aide à la création, il est également de coutume d'intégrer des possibilités d'import de données pour permettre la création de nombreux éléments dans une tâche courte. Les données de ces éléments sont peut-être déjà toutes présentes dans une autre source, ou l'utilisateur aura plus de faciliter de les définir dans un autre logiciel. Il nous faudra définir précisément les formats utilisés et leur contenu.

Afin d'assurer une cohérence des données, il est préférable de ne pas autoriser la création d'une relation dont l'un des nœuds n'existe pas encore dans le système. De plus, toute erreur rencontrée doit être transmise à l'utilisateur, et l'import interrompu voire annulé. Celui-ci doit donc se dérouler en deux étapes successives :

1. Import des données des éléments
2. Import des données des relations

Parmis les formats de fichiers d'import, nous pouvons mentionner les plus classiques : Excel, CSV, XML, JSON. Puisque nous manipulons essentiellement des graphes, nous pouvons également piocher dans les formats d'échange prévus pour ceux-ci : GraphML, GXL, GML et autres.

7.7.4 Exécution de scripts

Puisque nous autorisons l'utilisation de langages de requête (e.g. SQL, Cypher, etc.), nous pouvons également permettre l'exécution de scripts rédigés avec ceux-ci. Nous pouvons également ouvrir notre solution à une utilisation par ligne de commande et ainsi apporter toute les fonctionnalités à des scripts de type shell, bash, batch et consorts.

Troisième partie

Construction et Validation de la solution

Chapitre 8

Construction et validation d'un prototype

Afin de mettre à l'épreuve notre solution, nous avons développé un prototype logiciel et observé ce dernier dans un cas réel d'infrastructure de production cordialement mise à notre disposition.

Notre prototype couvre, à ce stade, les fonctionnalités suivantes :
- Récupération et identification des éléments auprès d'un système de surveillance (Nagios XI v3.5)
- Constitution d'une représentation de ces éléments comportant les différentes relations identifiées dans le chapitre 5
- Génération du graphe de dépendances correspondant
- Exécution d'analyses de causes racines et d'impacts
- Présentation et manipulation de la représentation dans une interface visuelle

Pour cette première version, nous avons choisi d'utiliser une architecture trois tiers et différentes librairies logicielles que nous décrirons ci-après.

8.1 Architecture

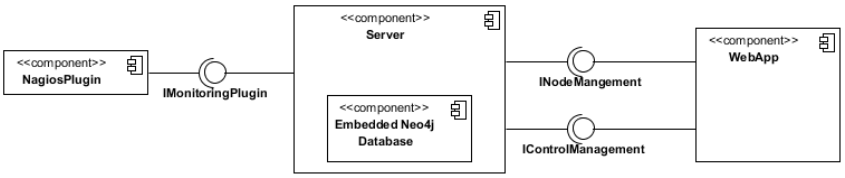

FIGURE 8.1 – Architecture du prototype

Nous décomposons notre architecture en trois composants essentiels : le serveur, l'application web (webapp) servant d'interface visuelle et le plugin adapté à Nagios.

8.2 Technologies et librairies

Le choix des technologies utilisées est principalement orienté par la connaissance et l'expérience que nous en avons déjà. Pour ce prototype, nous avons souhaité nous concentrer sur les fonctionnalités présentées plus que sur les possibilités ou les performances.

Le langage de programmation utilisé est le Java dans sa version 1.7, mais l'architecture décrite reste valable pour n'importe quel langage orienté objet. Nous utilisons le Spring Framework pour le composant serveur, Spring MVC pour l'application web et Spring Remoting pour le dialogue entre les deux.

Les échanges de données sont fait au format JSON, à l'aide de la librairie Google Gson. Le système de log sera celui de SLF4j. Nous exploitons également la librairie Apache Commons pour les manipulations de chaines de caractères et de fichiers.

Nous utilisons une base de données Neo4j que nous orchestrons à l'aide de Spring Data Neo4j et des librairies Java de Neo4j. Pour la représentation visuelle des graphes, nous utilisons la librairie Cytoscape.js et le framework css Twitter Bootstrap.

Enfin, le tout est compilé et construit à l'aide de Maven.

8.3 Composants

Afin d'améliorer la lisibilité des diagrammes présentés ci-dessous, nous masquerons les éléments qui ne nous intéressent pas directement, tels que les fonctions de démarrage et d'arrêt des applicatifs, les librairies et autres utilitaires.

8.3.1 Serveur

Nous divisons notre composant serveur en différents packages pour une maintenance aisée :

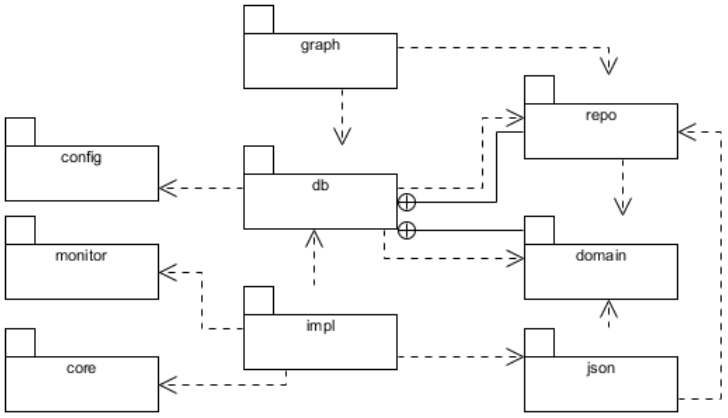

FIGURE 8.2 – Diagramme de packages - Serveur

db : contient le package *domain* comprenant les objets du domaine d'application (i.e. nœuds et relations) et *repo*, contenant le repository représentant l'ensemble du graphe présent en base de donnée.
impl : contient les implémentations des interfaces du serveur
monitor : contient la gestion des plugins d'accès aux systèmes de surveillance
core : gestion de l'applicatif (démarrage, arrêt, threads, état de fonctionnement,...)
config : reprend la configuration du serveur
graph : contient les composants d'analyse
json : gère les échanges de données

Les packages les plus intéressants sont détaillés dans le diagramme de classe de la figure 8.3. Les diagrammes de séquence des analyses sont présentés dans les figures 8.4, 8.5, 8.6 et 8.7.

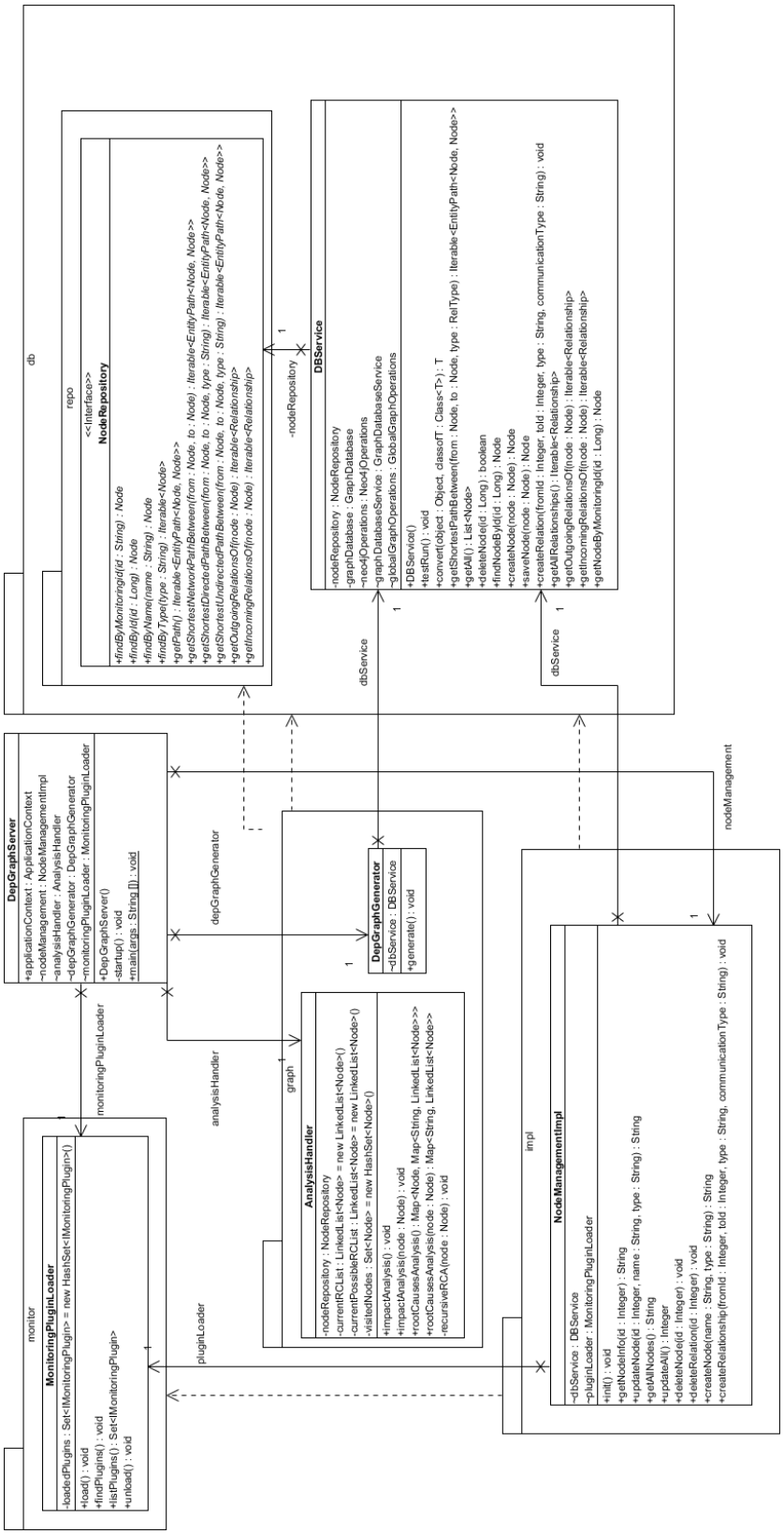

FIGURE 8.3 – Diagramme de classes - Serveur

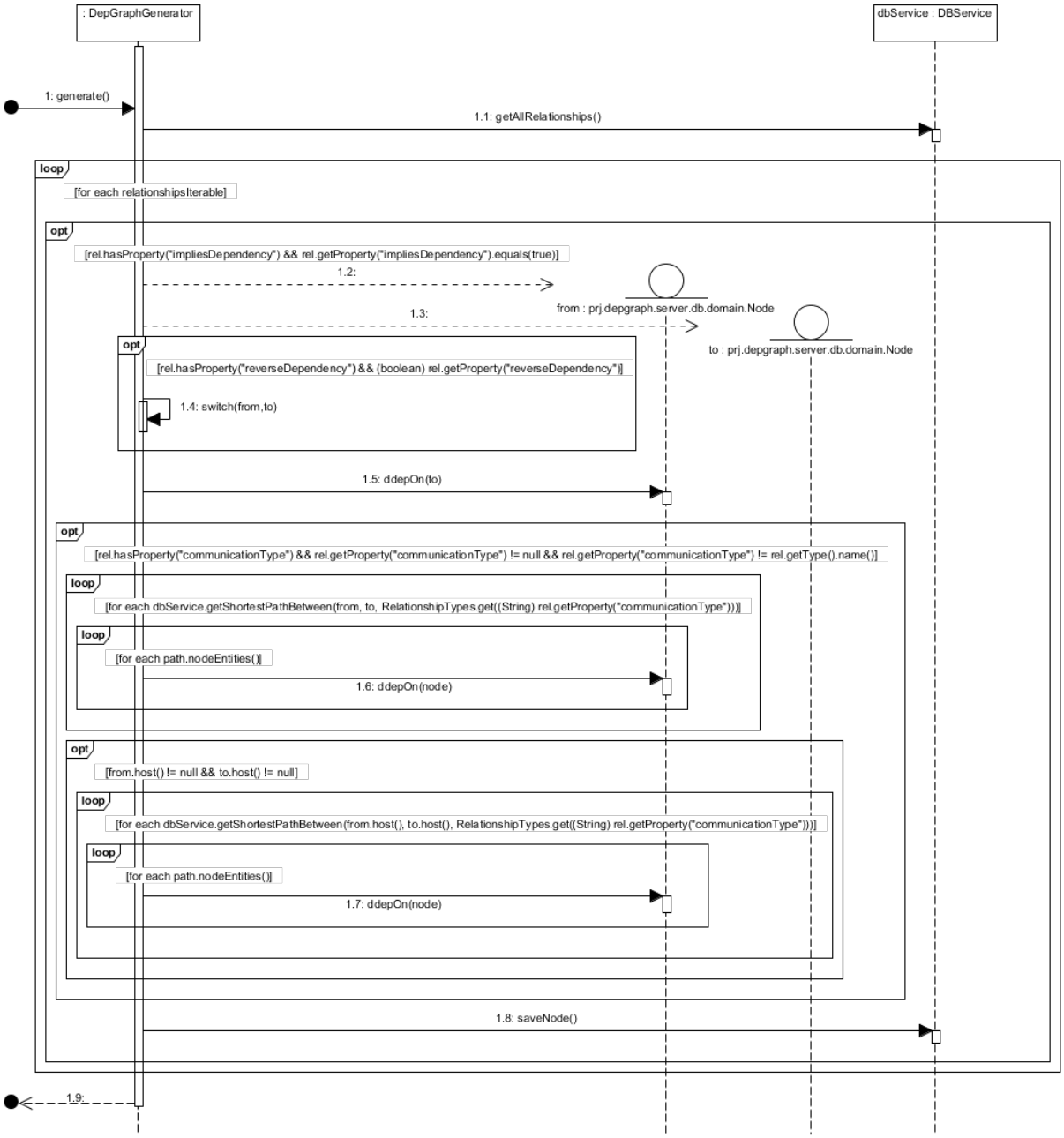

FIGURE 8.4 – Diagramme de séquence - Génération du graphe de dépendance

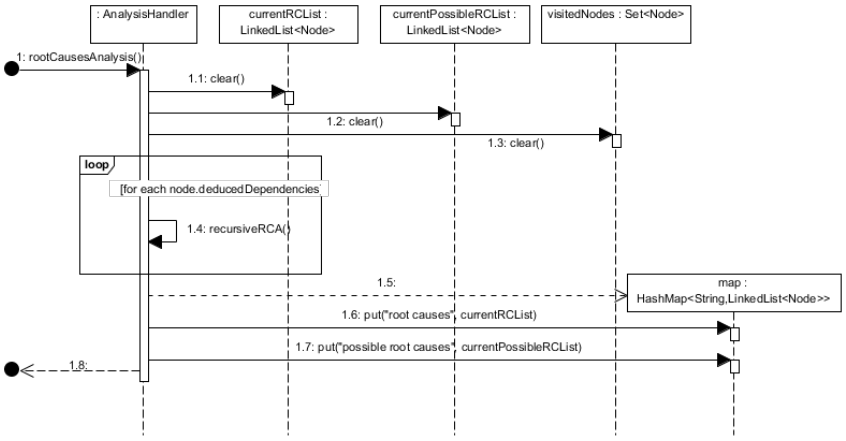

FIGURE 8.5 – Diagramme de séquence - Analyse de causes racines

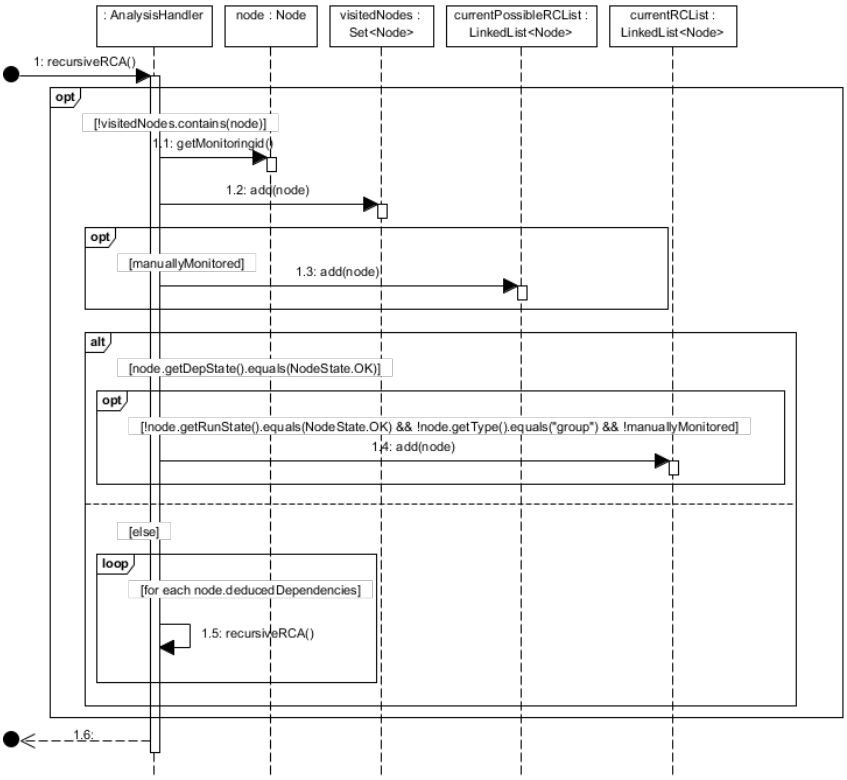

FIGURE 8.6 – Diagramme de séquence - Analyse de causes racines (récursion)

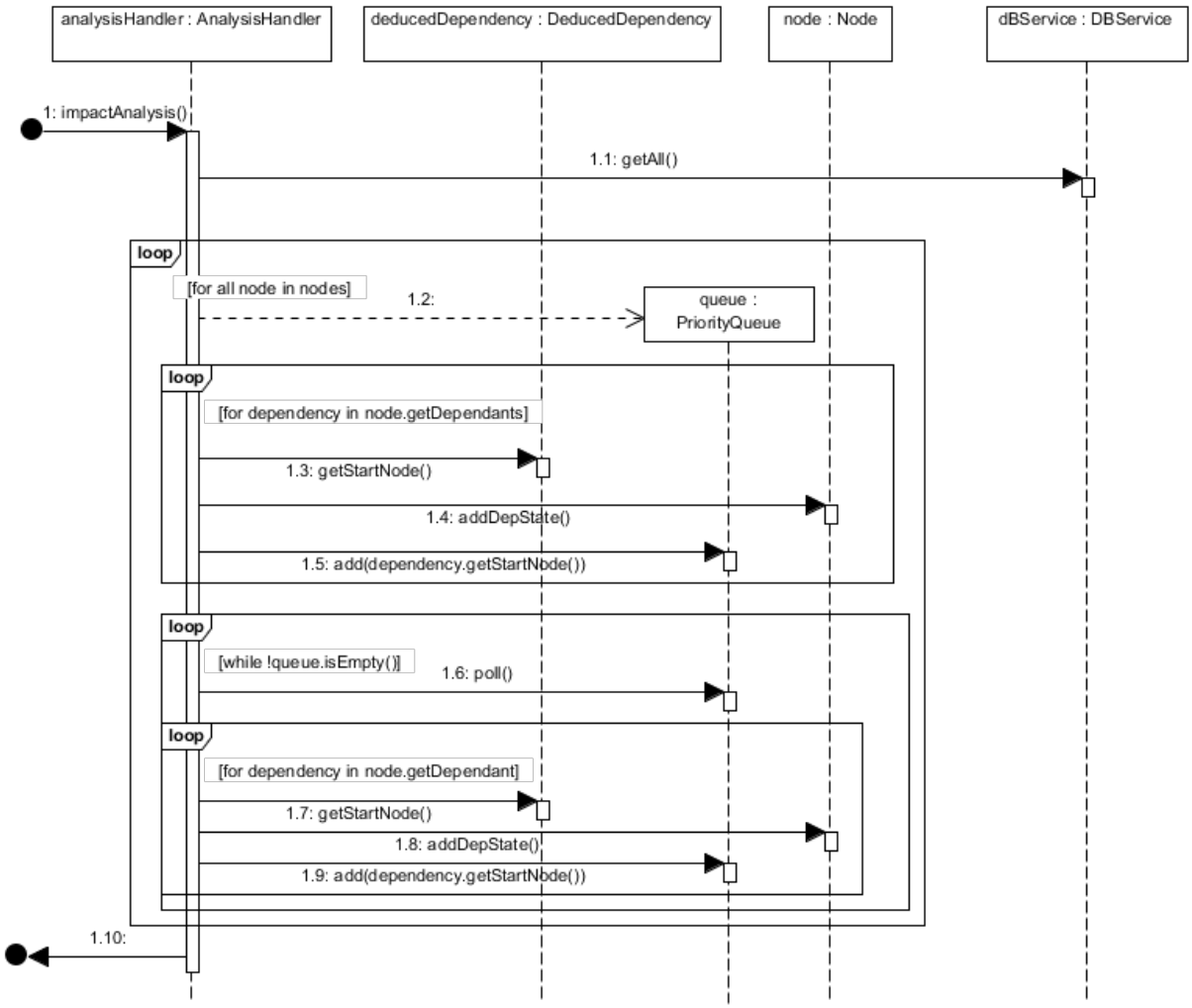

FIGURE 8.7 – Diagramme de séquence - Analyse d'impact

8.3.2 Plugin de connexion

Le plugin de connexion à Nagios utilise la Backend API d'un serveur Nagios qui fourni l'ensemble des données au format XML. Il doit contenir au moins une classe qui implémente l'interface IMonitorPlugin du serveur. Celle-ci sera chargée à l'aide d'un *URLClassLoader* de java. Les données des fichiers XML sont analysées à l'aide de Content Handler de la librairie SAX. Le diagramme de classe est présenté à la figure 8.8

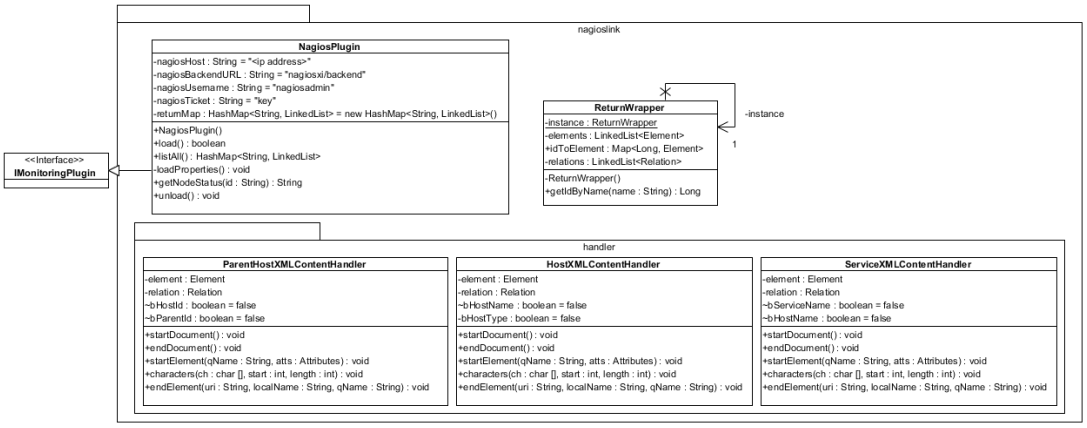

FIGURE 8.8 – Diagramme de classe - Plugin de connexion Nagios

8.3.3 Application web

L'application web est pour le moment une simple présentation du graphe et des données des nœuds et relations. L'interface est présentée à la figure 8.9. L'utilisation de la librairie Cytoscape.js nous permet d'avoir une représentation interactive que nous pouvons manipuler aisément.

8.4 Données

Le domaine d'application est contenu dans le package server.db.domain et contient l'objet Node ainsi que les différentes relations. Le diagramme de classe est présenté à la figure 8.10.

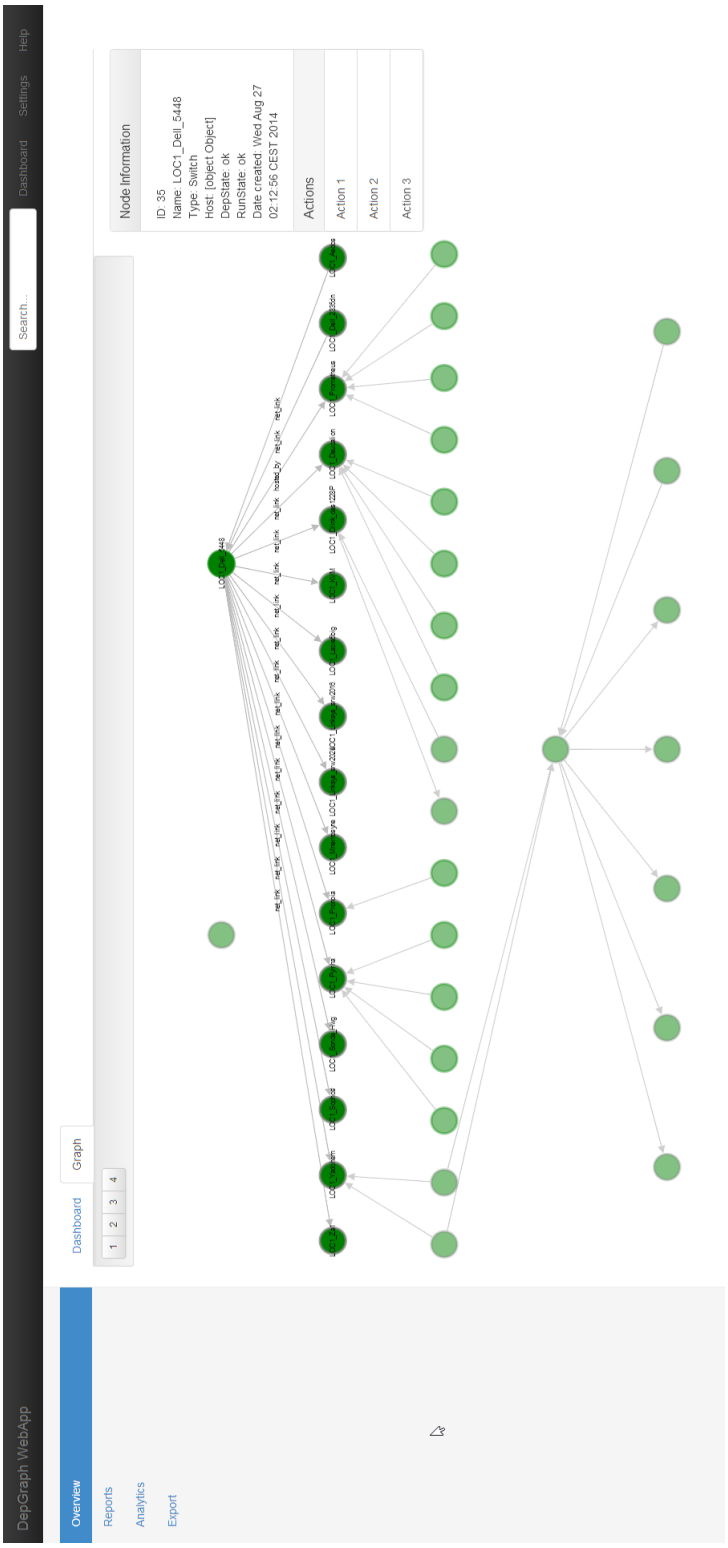

FIGURE 8.9 – Interface graphique - WebApp

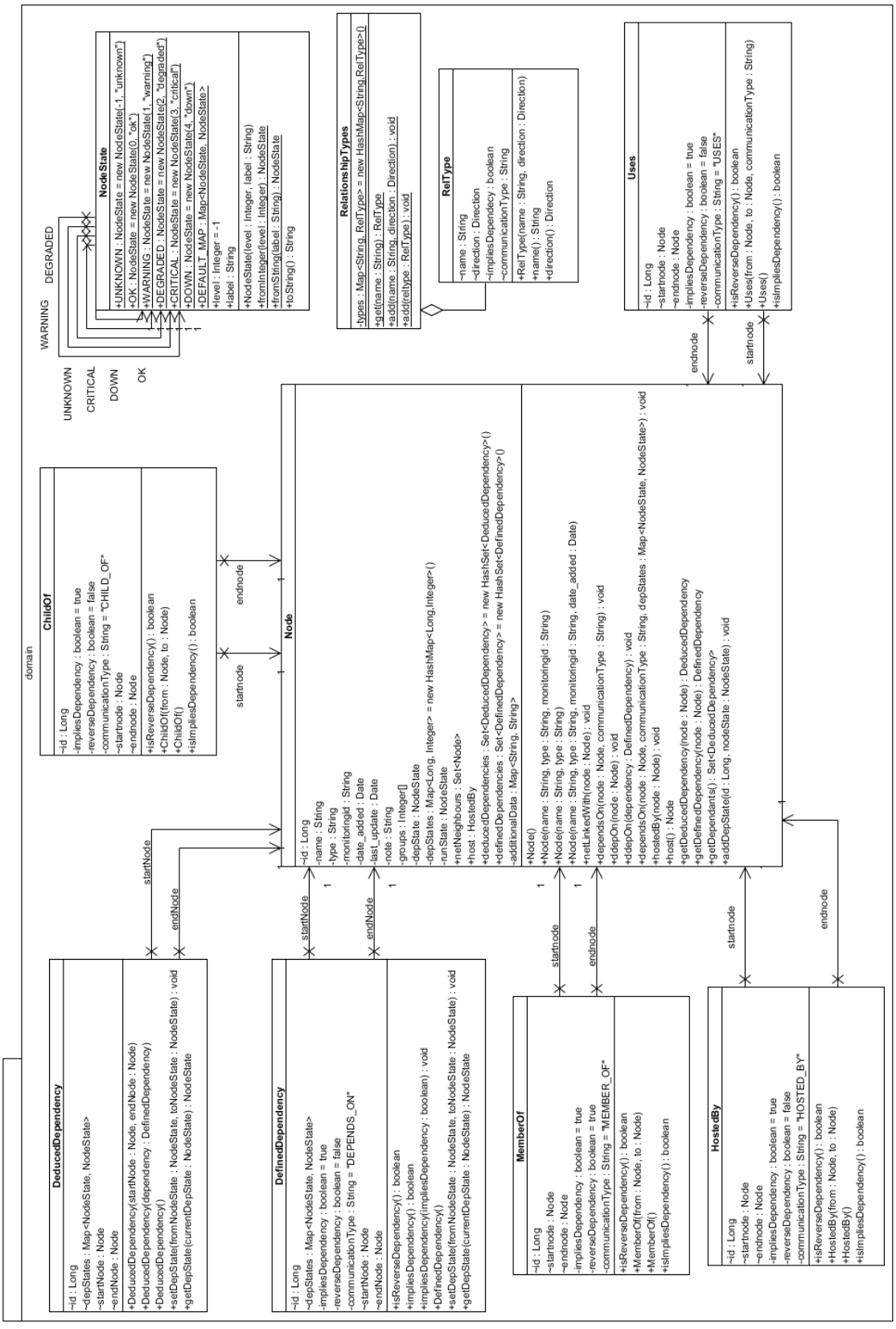

FIGURE 8.10 – Diagramme de classes - Domaine d'application

8.5 Validation

Nous avons pu éprouver notre prototype sur une infrastructure de production. L'impact sur les performances est quasiment nul puisqu'il s'agit ici simplement de dialoguer avec le système de surveillance et non les éléments de la salle machine. Dans un soucis de confidentialité, les noms des éléments ont été rendu anonymes.

8.5.1 Scénario de test

L'objectif de ce test était d'intégrer le prototype dans une infrastructure, le relier au système de surveillance et d'observer ce qu'il était capable de générer comme graphes. L'infrastructure concernée, telle que présentée par Nagios, était la suivante :

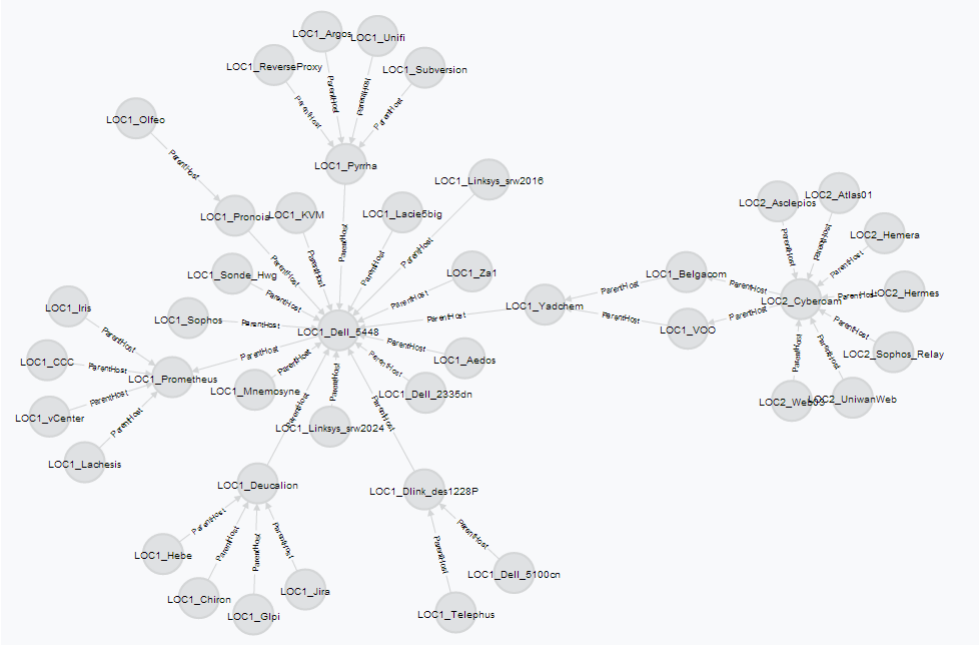

FIGURE 8.11 – Infrastructure de test

8.5.2 Scénarion réel

Lors des tests, nous avons rencontré plusieurs problèmes :

1. Nagios ne fait aucune distinction entre les serveurs physiques et les services logiciels.
2. Aucune relation n'y était définie. La seule notion présente est celle de "Parent" qui détermine l'inaccessibilité d'un hôte enfant si le parent est inaccessible

Nous avons donc été limité, dans cette version du prototype à l'identification des relations de types "HOSTED BY" et "NETWORK LINK" à l'aide des types des hôtes d'origine et de destination des relations "Parent". Le graphe que nous avons ainsi pu générer est néanmoins utilisable par notre prototype. La génération du graphe de dépendances a correctement relié les serveurs et leur hôte. L'analyse d'impact a pu établir l'état de dépendance de ces serveurs :

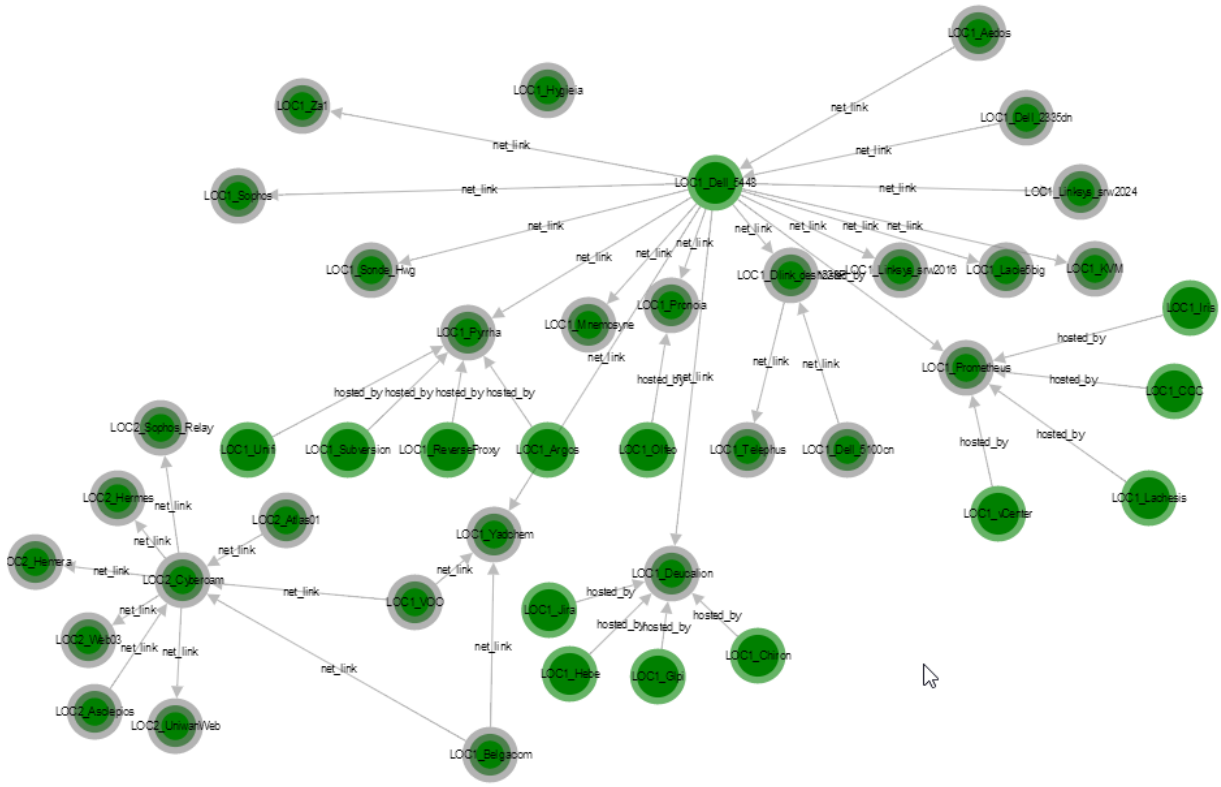

FIGURE 8.12 – Résultat des opérations du prototype

Pour les besoins du test de l'analyse de causes racines, il a été nécessaire de modifier directment les statuts de éléments dans les fichiers XML fournis par Nagios. Toutes les informations ont peut être retrouvées avec succès lors de l'analyse, qui n'est, somme toute que l'exécution d'un algorithme indépendant de l'infrastructure concernée.

Quatrième partie

Conclusion et Perspectives

Conclusion

Afin de répondre à la problématique de départ, il a d'abord été nécessaire la définir et d'analyser les tenants et aboutissants d'une telle question. Nous avons ensuite présenté les différents outils qui nous aiderait dans cette tâche. Après de longues et nombreuses heures de réflexion, nous avons abouti à une solution et l'avons présentée au long des chapitres de ce mémoire.

Ce travail de recherche, de conception et de développement nous a enseigné de nombreux concepts et théories. Il a ouvert nos horizons sur l'étendue des possibilités des systèmes de surveillance, sur les diverses recherches qui ont été menées sur l'identification automatique de relations entre composants.

Il a également élargi notre compréhension des systèmes d'information et des diverses façons qu'ils ont de cohabiter au sein d'une infrastructure. Il a également été question de rapprocher ces systèmes à des processus de gestion, de décision et de vision à long terme.

Il nous a permis d'apporter notre contribution à cette discipline.

Nous estimons avoir atteint un niveau de finissions qui répond à la problématique de départ. Cependant, afin de nous focaliser sur celle-ci, nous avons choisi d'ignorer un certain nombre d'aspect qu'une solution de ce type devrait comporter. Nous pensons aux aspects de sécurité, de confidentialité et d'intégrité de l'information, mais également aux performances et l'accessibilité de la solution.

Avec une connaissance et une expérience plus pointue des différents systèmes que l'on retrouve dans une salle machine, nous aurions pu approfondir et améliorer les possibilités d'identification des relations tant statiques que dynamiques. Nous espérons, avec le temps, pallier à ce manque. Si le temps nous l'avait permis, nous aurions également investigué la piste de l'intelligence artificielle pour une telle solution qui se doit de prendre conscience d'un ensemble complexe de diverses technologies pour perfectionner son utilité.

Perspectives

Outre le perfectionnement des fonctionnalités déjà présentes, nous entrapercevons plusieurs pistes d'évolution future et de domaine d'application.

Le premier domaine est bien entendu la gestion d'infrastructures informatiques, cependant, l'essence même d'une solution permettant de gérer des dépendances entre composants d'un même système nous permet aisément d'envisager l'application d'un principe identique à de nombreux domaines tels que le bâtiment, la finance, l'administration, la direction et bien d'autres.

L'application peut également être pédagogique au même titre que toute solution de modélisation qui permet d'expérimenter la théorie sur des exemples concrets à frais réduits.

Bibliographie

[Bagchi *et al.*, 2001] Bagchi, S., Kar, G. et Hellerstein, J. (2001). Dependency analysis in distributed systems using fault injection : Application to problem determination in an e-commerce environment.

[Bodenstaff *et al.*, 2008] Bodenstaff, L., Wombacher, A., Reichert, M. et Jaeger, M. C. (2008). Monitoring dependencies for slas : The mode4sla approach. *In Proceedings of the 2008 IEEE International Conference on Services Computing - Volume 1*, SCC '08, pages 21–29, Washington, DC, USA. IEEE Computer Society.

[Brown *et al.*, 2001] Brown, A., Kar, G. et Keller, A. (2001). An active approach to characterizing dynamic dependencies for problem determination in a distributed environment. *In Integrated Network Management Proceedings, 2001 IEEE/IFIP International Symposium on*, pages 377–390. IEEE.

[Colin, 2012] Colin, J.-N. (2012). *Sécurité et Fiabilité des Systèmes Informatiques.* Université de Namur.

[Dragich, 2012] Dragich, L. (2012). Event management : Reactive, proactive or predictive ? [http ://apmdigest.com/event-management-reactive-proactive-or-predictive ; accessed 28-April-2014].

[Ensel, 1999] Ensel, C. (1999). Automated generation of dependency models for service management. *In Workshop of the OpenView University Association (OVUA'99).*

[Gruschke *et al.*, 1998] Gruschke, B. *et al.* (1998). Integrated event management : Event correlation using dependency graphs. *In Proceedings of the 9th IFIP/IEEE International Workshop on Distributed Systems : Operations & Management (DSOM 98)*, pages 130–141.

[Himsolt, 2010] Himsolt, M. (2010). Gml : A portable graph file format. Rapport technique.

[Holt *et al.*, 2002] Holt, R., Schurr, A., Sim, S. E. et Winter, A. (2002). Graph exchange language. available at http ://www.gupro.de/GXL/, accessed 21-May-2014.

[Kar *et al.*, 2000] Kar, G., Keller, A. et Calo, S. (2000). Managing application services over service provider networks : architecture and dependency analysis. *In Network Operations and Management Symposium, 2000. NOMS 2000. 2000 IEEE/IFIP*, pages 61–74.

[Keller et Kar, 2001] Keller, A. et Kar, G. (2001). Determining service dependencies in distributed systems.

[Leclerq, 2010] Leclerq, J. (2010). *Théorie des Graphes.* Université de Namur.

[Microsoft, 2014] Microsoft (2014). Monitoring : Best practices. http ://msdn.microsoft.com/en-us/library/windows/desktop/bb226833accessed 30-April-2014].

[Nivre et McDonald, 2008] Nivre, J. et McDonald, R. T. (2008). Integrating graph-based and transition-based dependency parsers. *In ACL*, pages 950–958.

[Ortmeier *et al.*, 2006] Ortmeier, F., Reif, W. et Schellhorn, G. (2006). Deductive cause-consequence analysis (dcca). *In Proceedings of IFAC World Congress. Elsevier*. Citeseer.

[QualNet, 2013] QualNet (2013). Dossier n°61 : Les tableaux de bord. [http ://www.qualiteonline.com/rubriques/rub_3/dossier-61-les-tableaux-de-bord.html ; accessed 05-January-2014].

[Ramdoyal, 2012] Ramdoyal, R. (2012). *Interfaces Homme-Machine*. Université de Namur.

[Rizk et Ratajczak, 2012] Rizk, K. et Ratajczak, G. (2012). Failure mode and effects analysis (fmea) introductory overview.

[Robert Borgovini et Rossi, 1993] Robert Borgovini, S. P. et Rossi, M. (1993). Failure mode, effects, and criticality analysis (fmeca).

[USNRC, 1998] USNRC (1998). *Fault Tree Handbook U.S. Nuclear Regulatory Commission Nureg-0492.*

[Wikipedia, 2013] Wikipedia (2013). Network monitoring interface card. [http ://en.wikipedia.org/wiki/Network_Monitoring_Interface_Card ; accessed 26-April-2014].

[Wikipedia, 2014a] Wikipedia (2014a). Common management information protocol. [http ://en.wikipedia.org/wiki/Common_management_information_protocol ; accessed 26-April-2014].

[Wikipedia, 2014b] Wikipedia (2014b). Fault, configuration, accounting, performance, security. [http ://en.wikipedia.org/wiki/Network_management_model ; accessed 30-April-2014].

[Wikipedia, 2014c] Wikipedia (2014c). Management information base. [http ://en.wikipedia.org/wiki/Management_information_base ; accessed 26-April-2014].

[Wikipedia, 2014d] Wikipedia (2014d). Network monitoring. [http ://en.wikipedia.org/wiki/Network_management_system ; accessed 26-April-2014].

[Wikipedia, 2014e] Wikipedia (2014e). System monitoring. [http ://en.wikipedia.org/wiki/System_monitoring ; accessed 26-April-2014].

[Zimmermann et Nagappan, 2008] Zimmermann, T. et Nagappan, N. (2008). Predicting defects using network analysis on dependency graphs. *In Proceedings of the 30th International Conference on Software Engineering.*

Annexes

Annexe A

Exemples de formats de fichier pour les graphes

A.1 Graph Modelling Language - GML

[Himsolt, 2010]

A.1.1 Syntaxe

```
GML ::=List
List ::=(whitespace*Keywhitespace+Value)*
Value ::=Integer|Real|String|[List]
Key ::=[a-zA-Z][a-zA-Z0-9]*
Integer ::=signdigit+
Real ::=signdigit*.digit*mantissa
String ::="instring"
sign ::=empty|+|-
digit ::=[0-9]
Mantissa ::=empty|E sign digit
instring ::=ASCII-{&,"}|&character+;
whitespace ::=space|tabulator|newline
```

A.1.2 Exemple

```
graph [
        comment "This is a sample graph"
        directed 1
        id 42
        label "Hello, I am a graph"
        node [
                id 1
```

```
                label "node 1"
                thisIsASampleAttribute 42
        ]
        node [
                id 2
                label "node 2"
                thisIsASampleAttribute 43
        ]
        node [
                id 3
                label "node 3"
                thisIsASampleAttribute 44
        ]
        edge [
                source 1
                target 2
                label "Edge from node 1 to node 2"
        ]
        edge [
                source 2
                target 3
                label "Edge from node 2 to node 3"
        ]
        edge [
                source 3
                target 1
                label "Edge from node 3 to node 1"
        ]
]
```

A.2 GraphML

Les spécifications de syntaxe sont définies par la Document Type Definition (DTD) disponible à l'adresse `http://graphml.graphdrawing.org/xmlns/1.1/graphml.xsd`.

A.2.1 Exemple

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns
        http://graphml.graphdrawing.org/xmlns/1.0/graphml.xsd">
  <key id="d0" for="node" attr.name="color" attr.type="string">
    <default>yellow</default>
  </key>
```

```
  <key id="d1" for="edge" attr.name="weight" attr.type="double"/>
  <graph id="G" edgedefault="undirected">
    <node id="n0">
      <data key="d0">green</data>
    </node>
    <node id="n1"/>
    <node id="n2">
      <data key="d0">blue</data>
    </node>
    <node id="n3">
      <data key="d0">red</data>
    </node>
    <node id="n4"/>
    <node id="n5">
      <data key="d0">turquoise</data>
    </node>
    <edge id="e0" source="n0" target="n2">
      <data key="d1">1.0</data>
    </edge>
    <edge id="e1" source="n0" target="n1">
      <data key="d1">1.0</data>
    </edge>
    <edge id="e2" source="n1" target="n3">
      <data key="d1">2.0</data>
    </edge>
    <edge id="e3" source="n3" target="n2"/>
    <edge id="e4" source="n2" target="n4"/>
    <edge id="e5" source="n3" target="n5"/>
    <edge id="e6" source="n5" target="n4">
      <data key="d1">1.1</data>
    </edge>
  </graph>
</graphml>
```

A.3 Graphviz et langage DOT

A.3.1 Syntaxe

```
graph : [ strict ] (graph | digraph) [ ID ] '{' stmt_list '}'
stmt_list : [ stmt [ ';' ] [ stmt_list ] ]
stmt : node_stmt
       | edge_stmt
       | attr_stmt
       | ID '=' ID
       | subgraph
```

```
attr_stmt : (graph | node | edge) attr_list
attr_list : '[' [ a_list ] ']' [ attr_list ]
a_list : ID '=' ID [ (';' | ',') ] [ a_list ]
edge_stmt : (node_id | subgraph) edgeRHS [ attr_list ]
edgeRHS : edgeop (node_id | subgraph) [ edgeRHS ]
node_stmt : node_id [ attr_list ]
node_id : ID [ port ]
port : ':' ID [ ':' compass_pt ]
       | ':' compass_pt
subgraph : [ subgraph [ ID ] ] '{' stmt_list '}'
compass_pt : (n | ne | e | se | s | sw | w | nw | c | _)
```

A.3.2 Exemple

```
digraph TrafficLights {
node [shape=box];  gy2; yr2; rg2; gy1; yr1; rg1;
node [shape=circle,fixedsize=true,width=0.9];
green2; yellow2; red2; safe2; safe1; green1; yellow1; red1;
gy2->yellow2;
rg2->green2;
yr2->safe1;
yr2->red2;
safe2->rg2;
green2->gy2;
yellow2->yr2;
red2->rg2;
gy1->yellow1;
rg1->green1;
yr1->safe2;
yr1->red1;
safe1->rg1;
green1->gy1;
yellow1->yr1;
red1->rg1;

overlap=false
label="PetriNet Model TrafficLights\n
Extracted from ConceptBase and layed out by Graphviz"
fontsize=12;
}
```