Software Asset Management and Cloud Computing

Anne-Lucie Vion

Communauté
UNIVERSITÉ Grenoble Alpes

THÈSE

Pour obtenir le grade de

DOCTEUR DE LA COMMUNAUTE UNIVERSITE GRENOBLE ALPES

Spécialité : **Informatique**

Arrêté ministériel : 25 mai 2016

Présentée par

Anne-Lucie Vion

Thèse dirigée par **Noël DE PALMA, Professeur à l'Université Joseph Fournier** et
Codirigée par **Fabienne BOYER, Professeur à l'Université Joseph Fournier**

Préparée au sein du département **Orange Labs Services** et du **Laboratoire d'Informatique de Grenoble**
dans **l'École Doctorale Mathématiques, Sciences et Technologies de l'Information, Informatique**

Software Asset Management & Cloud Computing

Thèse soutenue publiquement le **29 mars 2018**
devant le jury composé de :

M. Eddy CARON
Maître de conférences HDR à l'Ecole Normale Supérieure de Lyon (Président du jury)
Mme. Laurence DUCHIEN
Professeur à l'Université de Lille (Rapporteur)
M. Daniel HAGIMONT
Professeur à l'Institut National Polytechnique de Toulouse (Rapporteur)
Mme. Fabienne BOYER
Maître de conférences à l'Université de Grenoble-Alpes (Co-directrice de thèse)
M. Noël DE PALMA
Professeur à l'Université de Grenoble-Alpes (Directeur de thèse)
Mme. Noëlle BAILLON
Responsable du programme SAM Orange (Co-Directrice de thèse)

Pour Zoé, j'ai enfin fini mon « livre »,

Pour Félix qui a peur des docteurs,

Pour L. et tous les bonheurs à venir.

REMERCIEMENTS

En tout premier lieu, je souhaite remercier l'ensemble des membres du jury qui m'ont fait l'honneur de participer à ma soutenance et consacré un temps précieux à la lecture de ce document. Tout particulièrement Daniel Hagimont et Laurence Duchien (rapporteurs) et Eddy Caron, (examinateur).

Je remercie très chaleureusement mes encadrants de thèse pour la confiance qu'ils m'ont accordée et leurs précieux conseils : Merci à Pascal Dechamboux d'avoir accepté d'encadrer ma thèse chez Orange. Merci à Noël De Palma, professeur à l'Université Joseph Fournier ainsi qu'à Mme Fabienne Boyer, professeur à l'Université Joseph Fournier. Un merci tout particulier à Noëlle Baillon-Bachoc, mon encadrante chez Orange pour sa présence bienveillante et ses encouragements quotidiens.

Merci aux équipes SAM et SUPRA pour leur accueil chaleureux, pour leur écoute attentive et la qualité des échanges que nous avons eu. Merci à Karl pour son support technique et sa patience face à mes questions parfois triviales.

Je remercie du fond du cœur mes parents qui ont fait grandir en moi la soif d'apprendre ; ma famille pour les joies partagées et toutes celles que nous partagerons encore, mon compagnon Janusz pour n'avoir jamais douté de ma capacité d'aller jusqu'au bout malgré les embûches.

Enfin, mon plus grand merci est adressé à Zoé et Félix, leur patience et l'amour inconditionnel pour leur maman parfois irritable ont été ma plus grande force dans l'accomplissement de ce projet.

RÉSUMÉ

Dans le Cloud, peu de travaux traitent de l'analyse de l'usage réel et dynamique des logiciels consommés afin de déterminer les coûts réels engendrés et le respect des droits acquis auprès des fournisseurs de ces ressources. L'émergence de la pratique du Software Asset Management (SAM) traduit pourtant la préoccupation grandissante des industriels et des 'Telcos' (Entreprises de télécommunications) face à la complexité des modèles de licences dans des environnements virtualisés qui bouleversent nos usages de logiciel.

La réponse des éditeurs de logiciel est souvent une incitation à ne plus suivre la consommation de licences, par le biais de contrats onéreux de consommation illimitée, rendant impossible une politique de maîtrise des coûts. Pour les utilisateurs finaux comme pour les fournisseurs de services cloud, il devient impératif de maîtriser et d'optimiser le déploiement des licences dans le Cloud.

L'objectif devient celui de maitriser les besoins logiciels, au plus proche du temps réel, puis de générer des scénarii d'optimisation basés sur l'évolution de la consommation en modélisant les coûts réels afférents. Cela représente un levier de gains considérables pour tous les acteurs du cycle de vie du logiciel.

Le contexte d'étude couvre l'ensemble du scope du Cloud (applications, plateformes, infrastructures et réseaux). Les travaux présentés ici s'attachent à reconstituer tout le cycle de vie du logiciel, de l'achat jusqu'à la désinstallation, en intégrant les contraintes liées à sa nature ou à son usage. Nous proposons de résoudre le verrou majeur de l'identification du logiciel et de ses droits d'usage par la création et le suivi d'un tag.

Nous proposons également une modélisation innovante s'appuyant sur une base de données graphe qui permet d'intégrer l'instantanéité des changements de configuration, de prendre en compte les différentes responsabilités impliquées par les niveaux de services offerts, tout en offrant la souplesse nécessaire pour supporter à la fois des modèles de licence classiques, ou à l'usage.

Deux cas d'usages seront envisagés pour juger de la pertinence des modèles proposés : la gestion des licences dans un contexte de Plateforme as a Service (PaaS) et dans un cas de virtualisation de réseau (NFV).

ABSTRACT

About Cloud, only few works deals with dynamic and real usage analysis of deployed software in order to determine the true related costs, and licensing compliance with acquired rights from the software editors.

However, the emergence of Software Asset Management (SAM) shows the growing concerns of the industry and carriers facing the licensing model complexity especially in virtualized environments where the software usage is disrupted.

Editor's answer consists in proposing to stop following this consumption via very expensive unlimited-usage contracts. It makes impossible to implement true cost management policies. For final users like for cloud service providers, it is crucial to manage and optimize license deployments in cloud environments.

Firstly the aim is to control Software needs, as close as possible to real time, then to generate optimization scenarios based on consumption evolution by cost modeling.

It represents a valuable saving leverage and may let spring up new licensing models, more profitable for software lifecycle's stakeholders.

Usage context covers all scope of Cloud (application, infrastructure and network). Our works propose to rebuild the Software life-cycle, from procurement to uninstallation, encompassing the constraints of it nature and usages. We propose to solve software identification issue by creation and monitoring of tags.

Additionally, we propose an innovative modeling based on a graph database which allows instant integration of configuration changes, to take into account the different levels of responsibility induced by the different levels of granted services. It offers enough flexibility to handle classical licensing models as use-based model which are often more attractive for cloud-users.

Two use-cases will be developed to evaluate our model efficiency: the software licensing management in PaaS (Platform as a Service) context and in NFV environment (Network Function Virtualization).

TABLE OF CONTENTS

Terms, Definitions and abbreviated terms 19

I. Introduction 26

1. Cloud Computing Overview 27

2. Software Financial Issues: Audits And Wastes On The Rise 29

3. The necessary emergence of Software Asset Management 31

4. New Factors affecting Software and SAM new paradigm 33

5. Contributions and followings 34

II. State of the art 37

1. Academic State of The Art 38

1.1. Theoretical works about SAM and its implementation limits 38

1.2. Nowadays context justifies emergence of SAM in the literature 41

1.3. Synthesis: SAM maturity scale 42

2. Industrial State of The Art 44

2.1. Requirements 44

2.2. Evaluation grid 44

2.3. Choice of SAM Market tools 49

2.4. Results of the evaluation 49

2.5. Synthesis on Industrial State of the Art 69

3. Synthesis 70

III. Proposition of a Software Identification Model for Cloud Environments 73

1. Software licensing issues, challenges and opportunities 77

1.1. Licensing complexities moving to the cloud 78

1.2. Cloud-oriented software licensing models 81

2. Requirements for effective identification of Software and its entitlements 81
2.1. Requirements for Software identification 82
2.2. Requirements for Usage Rights (PUR) or entitlements identification 82
2.3. Requirements for Instances and bounded deployment environment identification 84
2.4. Requirements for identification inherent processes 84
3. Software PUR management process flow 85
4. Implementation of software identification patterns 88
4.1. Software identification hindrances 88
4.2. Software identification models proposed by ISO 19770-x 90
5. Workaround propositions 98
5.1. About PUR 98
5.2. About SKU 99
5.3. Proposed identification Lifecycle 101

IV. Proposition of a SAM Model for the Cloud 105
1. SAM Control loop 106
1.1. Autonomic computing and general concept of control loop 106
1.2. Application to SAM Control Loop 107
1.3. SAM model 110
2. Database model for SAM loop 121
2.1. Relational Databases vs Graph Databases 121
2.2. From relation to Graph Databases 122
2.3. SAM Graph Proposition 125

V. Model Assessments 130
1. Platform Use Case I: PaaS - Cloud Foundry 132
1.1. Cloud- PaaS instantiation and usage capture 132
1.2. How does Cloud Foundry work? 133
1.3. Instances 134

1.4. Usage collect 135
1.5. Modeling 137
2. Platform Use Case II: Network Function Virtualization 157
2.1. Orchestration & Hypervisor in Operator's network..... 157
2.2. Context concerns..... 162
2.3. Usage collection 165
2.4. Modeling 165

VI. Conclusion..... 181
1. Reminding the issues.....181
2. Reminding the contributions 182
3. Further works.....183

Referenced works 185

TABLE OF FIGURES

Figure 1. Enterprise Cloud services spending forecast 2016-21 ($bn) 29
Figure 2 - Proposition for a SAM maturity scale 43
Figure 3 - Study's criteria weight 48
Figure 4 - Aspera Evaluation Summary 51
Figure 5 - Aspera Evaluation detailed view 52
Figure 6 - Snow Software Evaluation Summary 54
Figure 7 - Snow Software Evaluation detailed view 55
Figure 8 - Flexera Evaluation summary 57
Figure 9. Flexera Evaluation Detailed view 58
Figure 10 - Spider Evaluation Summary 59
Figure 11 - Spider Evaluation Detailed view 60
Figure 12 - Eracent Evaluation Summary 61
Figure 13 - Eracent Evaluation detailed view 62
Figure 14 - HP Asset Manager Evaluation Summary 63
Figure 15 - HP Asset Manager Evaluation detailed view 64
Figure 16 - BMC Remedy Evaluation Summary 65
Figure 17 - BMC Remedy Evaluation detailed view 66
Figure 18 - GLPI/OCSng Evaluation Summary 67
Figure 19 - GLPI/OCSng Evaluation detailed view 68
Figure 20 - Features and limitations of most popular SAM tools 69
Figure 21 - Complexity factors brought about cloud architecture 70
Figure 22 - Key operational processes in software PUR identification 87
Figure 23 - Recognition vs Identification 89
Figure 24 - SWID Tag lifecycle described in ISO 19770-2 93
Figure 25 - Analogy between PUR and railway 99
Figure 26 - Analogy between SKU and Juice Bottles 100
Figure 27 - Combination of SWID and SKU 101
Figure 28 - SWIDTag+ lifecycle with initial SWIDTag 102
Figure 29 - SWIDTag+ lifecycle without initial SWIDTag 102
Figure 30 - Software identification lifecycle from provisioning to billing ... 103
Figure 31 - Compliancy control loop 109
Figure 32 - Over-deployment control loop 109
Figure 33 - SAM lifecycle 110
Figure 34 - Software Lifecycle - Need & Purchase 113
Figure 35 - Software Lifecycle – Delivery 113
Figure 36 - Software Lifecycle – Instantiation 114
Figure 37 - Software Lifecycle – Usage 115

Figure 38 - SAM general retroaction loop and control 116
Figure 39 - Different measures of uses translated into licensing models 120
Figure 40 - Example of junction table to match people and project 122
Figure 41 - Example of graph linking a person with projects 123
Figure 42 - Graph Database 125
Figure 43 - SAM Graph Model 126
Figure 44 - Use case of Cloud App Access 136
Figure 45 - Usage metering and aggregation for Cloud Foundry 137
Figure 46 - Cloud Architecture Model 138
Figure 47 - Product Catalog 140
Figure 48 - Neo4J interface - Graph Step 1 141
Figure 49 - Neo4J interface - Graph Step 2 142
Figure 50 - Neo4J interface - Graph Step 3 143
Figure 51 - Neo4J interface - Graph Step 4 144
Figure 52 - Neo4J interface - Graph Step 5 145
Figure 53 - Neo4J interface - Graph Step 6 147
Figure 54 - Neo4J interface - Graph Step 7 148
Figure 55 - Neo4J interface - Graph Step 8 149
Figure 56 - Neo4J interface - Query Bought 150
Figure 57 - Neo4J interface - Query Instance 151
Figure 58 - Neo4J interface - Query Compliance 151
Figure 59 - Popoto for graphic Neo4J interface 152
Figure 60 - cSAM tool Features 153
Figure 61 - Asynchronous feeding of graph for Software lifecycle 154
Figure 62- cSAM - Simulation on Oracle DB licensing 156
Figure 63 - cSAM - Simulation on Oracle DB instance's resources 157
Figure 64 - HW and SW disconnection and separate lifecycle management 158
Figure 65 - NFV complexity factors for SAM 159
Figure 66 - License & metering server management model 161
Figure 67 - NFV Cloud Orchestration 163
Figure 68 - Blue Planet UI 164
Figure 69 - NFV Architecture Model 167
Figure 70 - Product Catalog (2) 168
Figure 71 - Neo4J interface – Graph 2 Step 1 169
Figure 72 - Neo4J interface – Graph 2 Step 2 170
Figure 73 - Neo4J interface – Graph 2 Step 3 172
Figure 74 - Neo4J interface – Graph 2 Step 4 173
Figure 75 - Neo4J interface – Graph 2 Step 5 174
Figure 76 - Neo4J interface – Graph 2 Step 6 175
Figure 77 - Neo4J interface – Graph 2 Step 7 177

Figure 78 - cSAM - simulate metric change 178
Figure 79 - cSAM - simulate resource change 179

Tables

Table 1. Remarkable figures quoted from Flexera survey about Software license audits in 2014 31
Table 2 - SAM Maturity Items 45
Table 3 - Most used metrics and identified risks in cloud environments 80
Table 4 - Some ECA model rules 108

TERMS, DEFINITIONS AND ABBREVIATED TERMS

Bundle

Grouping of products which is the result of a marketing/licensing strategy to sell entitlements to multiple products as one purchased item. A bundle can be referred to as a "suite", if the products are closely related and typically integrated(such as an office suite containing a spreadsheet, word processor, presentation and other related items). Bundles can also refer to software titles that are less closely related such as a game, a virus scanner and a utility "bundled" together with a new computer, or to groups of entitlements, such as multiple entitlements for a backup software product. [SOURCE: ISO/IEC 19770‑5, 3.5]

Customer

Organization or person that receives a product or service. [SOURCE: ISO/IEC 19770‑5, 3.10]

Downgrade right

Right granted to receive, install, and/or use an installation of a previous version of software than the currently granted entitlement. [SOURCE: ISO/IEC 19770‑5, 3.11]

Effector

An interface that enables state changes for a managed resource

End-user

Person or persons who will ultimately be using the system for its intended purpose. [SOURCE: ISO/IEC 19770‑5, 3.13]

Entitlement schema - Software entitlement schema - Ent

Information structure containing a digital encapsulation of a licensing transaction and its associated entitlement information. A single transaction does not necessarily encapsulate a full (or effective) entitlement. An effective entitlement may need to be determined by an analysis of multiple licensing transactions, of a full license and then of upgrades and/or maintenance transactions assessed together with it. [SOURCE: ISO/IEC 19770‑3]

Ent library – Service library

Construct which holds data about multiple Ents. The Ent library is typically a database, but could also be a file or other data storage mechanism. [SOURCE: ISO/IEC 19770-3]

Extensible markup language - XML

License-free and platform-independent markup language that carries rules for generating text formats that contain structured data. [SOURCE: W3C Recommendation Extensible Markup Language (XML) 1,1 (Second Edition), 1,2]

IT Asset Management

All the physical, logical, and virtual system platform, operating system, and software configuration information required for life cycle management of IT Assets. Asset Management systems compile accurate data about the IT environment, including the supporting resources (people, applications, infrastructure and information) and dependent services. Asset Management tracks and integrates the physical, logical, and virtual location of IT Service Assets with key financial properties. This collection of systems is focused on establishing a framework for managing service assets in an operational context. [SOURCE: Laura Knapp, IBM Services Management, The IBM® Software Group Strategy NoteBooks, 2008 http://w3-103.ibm.com/software/xl/portal/viewcontent?type=doc&srcID=XT&docID=L107895Y49377G53]

License model

Class of licenses with common characteristics. [SOURCE: ISO/IEC 19770-5]

Limit

Restriction on rights or privileges granted by a software entitlement

Original equipment manufacturer license

Oem license

License for products or components that are created or manufactured by one company and licensed by another company

Perpetual license

License for a software entitlement granted in perpetuity. The alternative to a perpetual license is a term or subscription-based license.

Software entitlement reconciliation

Process of comparing software entitlements owned with those required (granted and deployed), usually to determine compliance with software license agreements release collection of one or more new or changed configuration items deployed into the live environment as a result of one or more changes. [SOURCE: ISO/IEC 19770‑5, 3.28]

Right

Privilege or benefit granted by a software entitlement

SAM practitioner

Individual involved in the practice or role of managing software assets. A SAM practitioner is often involved in the collection or reconciliation of software inventory and/or software entitlements.[SOURCE: ISO/IEC 19770‑5, 3.31]

SAM tool

Software used to assist in and automate parts of the process of management of software assets

Sensor

An interface that exposes information about the state and state transitions of a managed resource.

Software

All or part of the programs, procedures, rules, and associated documentation of an information processing system. There are multiple definitions of software in use. For the purpose of this part of ISO/IEC 19770, it is typically important to include both executable and non-executable software, such as fonts, graphics, audio and video recordings, templates, dictionaries, documents and information structures, such as database records. [SOURCE: ISO/IEC 24765:2010, 3.34]

Software Asset Management

SAM

control and protection of software and related assets within an organization, and control and protection of information about related assets which are needed in order to control and protect software assets. For reference, a corresponding industry definition is “all of the infrastructure and processes necessary for the effective management, control and protection of the software assets within an organization, throughout all stages of their lifecycle”. [SOURCE: ISO/IEC 19770‑5, 3.35]

Software License Optimization

SLO

All actions enabling organizations to gain visibility and control of IT assets, reduce ongoing software costs, and maintain continuous license compliance.

Software creator

Person or organization that creates a software product or package. This entity might or might not own the rights to sell or distribute the software. [SOURCE: ISO/IEC 19770‑5, 3.38]

Software entitlement

Entitlement

Software license use rights as defined through agreements between a software licensor and a software consumer. Effective use rights take into account any contracts and all applicable licenses, including full licenses, upgrade licenses and maintenance agreements. [SOURCE: ISO/IEC 19770‑5, 3.39]

Software identification tag

SWID tag

SWID

Set of structured data elements containing authoritative identification information about a software configuration item. [SOURCE: ISO/IEC 19770‑2, 3.40]

Software license

Legal rights to use software in accordance with terms and conditions specified by the software licensor. “Using a software product” can include: accessing, copying, distributing, installing and executing the software product, depending on the license’s terms and conditions. [SOURCE: ISO/IEC 19770‑5, 3.41]

Software maintenance

Entitlement of additional rights (such as additional functionality, upgrade or support) for a previously granted software entitlement

Software package

Complete and documented set of software supplied for a specific application or function. In the iso/iec 19770 family of standards, the term software package refers to the set of files associated with a specific set of business functionality that can be installed on a computing device and has a set of specific licensing requirements. In the iso/iec 19770 family of standards, the terms “product” and “software package” are used synonymously depending on the context of the item described.

Software product

Complete set of software designed for delivery to a software consumer or end-user that may contain computer programs, procedures and associated documentation and data. In the ISO/IEC 19770 family of standards, the terms “software product” and “software package” are used interchangeably depending on the context of the item described. [SOURCE: ISO/IEC 19770‑5, 3.46]

Stock keeping unit

SKU

Identification, usually alphanumeric, of a particular product that allows it to be tracked for inventory and software entitlement purposes. The term “stock keeping unit” is traditionally associated with physical goods. In the sense of licenses it refers to a unique identifier, sometimes also called “part number”. The term “stock keeping unit” is typically associated with unique products for sales purposes, such as software entitlements. It may not correspond uniquely to specific software products, but may instead represent packages of software, and/or specific terms and conditions related to software products, such as whether it relates to a

full product, upgrade product, or maintenance on an existing product. [SOURCE: ISO/IEC 19770-1, 3.48]

Subscription-based license

Term-based license

Service-based license

License for an entitlement that is for a limited amount of time. This type of license shall be renewed to remain in force. Specifically it is not a perpetual license.

Chapter 1

I. INTRODUCTION

Introduction 26

1. Cloud computing overview 27
2. Software financial issues: Audits And wastes on the rise 29
3. The necessary emergence of Software Asset Management 31
4. New factors affecting software and SAM new paradigm 33
5. Contributions and followings 34

Cloud computing is on the rise as Software market struggle. Pricing and licensing systems become more and more complex and less and less understandable for the clients. To face market's growth stabilization, editors need to find new sources of income. The current economic climate underlines this particularly burning issue, as each non-compliance situation is heavily penalized in financial aspects. Therefore, we are seeing a rise of software compliance audits along with creation of dedicated unit showing their growing importance for some editors.

1. Cloud Computing Overview

Cloud computing is an information technology (IT) paradigm, a model for enabling ubiquitous access to shared pools of configurable resources. These resources can be rapidly provisioned with minimal management effort, often over the Internet. Computing relies on sharing of resources to achieve coherence and economy of scale.

Cloud computing comes in three forms: public clouds, private clouds, and hybrids clouds.

- Public clouds are based on shared physical hardware, owned and operated by a third-party provider. The main benefits of the public cloud encompass the speed of IT resources deployment and the alleged ability to pay only of the resources you use. The sheer size of public clouds allows scaling compute power up and down as business demands, within a matter of minutes.
- Private clouds are infrastructures dedicated entirely to their owner's business. They are hosted either on-site or in a service provider's data center. The private cloud delivers all the agility, scalability and efficiency of the public cloud, but also provides greater levels of control and security. A major benefit of private cloud is the ability to customize it components to best suit any specific IT requirements (something that cannot be achieved so easily in the public cloud environment).
- Hybrid clouds allow combining public cloud with private cloud or dedicated hosting and leverage the best of what each has to offer. For example, to use the public cloud for non-sensitive operations, the private cloud for business-critical operations, and incorporate any

existing dedicated resources to achieve a highly flexible, highly agile and highly cost-effective solution.

The cloud computing paradigm proposes the on-demand usage of provided and maintained resources on hardware and software level. The terms Infrastructure, Platform and Software as a Service (IaaS, PaaS and SaaS) sort three different service models and are widely used and commonly accepted in literature. Theoretically, they characterize different layers of abstraction at which cloud resources are offered at.

- IaaS is commonly perceived as providing resources on hardware level.
- PaaS allows tenants to deploy applications in a cloud environment.
- SaaS is the provisioning of whole applications as a resource.

The similarity between all cloud offerings is the provisioning of resources in a flexible and abstracted way. Literature identifies three important types of resource domains. Most prominent, computational resources allow the deployment, execution and use of software, it provide mechanisms to run applications. Besides, cloud systems may provide storage (offer a way to store data persistently) and network services (comprise any mechanism used to communicate between (virtual) machines, applications and users) usable either stand-alone or in conjunction with computational resources.

Cloud computing is increasingly being adopted by enterprises. According to Ovum[1], enterprise cloud services spending will grow at a CAGR[2] of 17.5% during the 2016–21 forecast periods[3] (See Fig 1). Within enterprise cloud, the SaaS market will remain dominant even in 2021, accounting for $97bn in global spend (a bit more than half the market). Platform-as-a-service (PaaS) will be the fastest-growing service line, with a CAGR of 29.6%.

[1] Ovum is a market-leading research and consulting business focused on helping digital service providers and their vendor partners thrive in the connected digital economy.

[2] Compound annual growth rate (CAGR) is the mean annual growth rate of an investment over a specified period of time longer than one year.

[3] Ovum Research, Demystifying Accounting for Software Expenses, 2017, Publication Date 26 Jun 2017, Product code: TE0006-001409, Analyst: Gaurav (Shukla, 2017) [1]

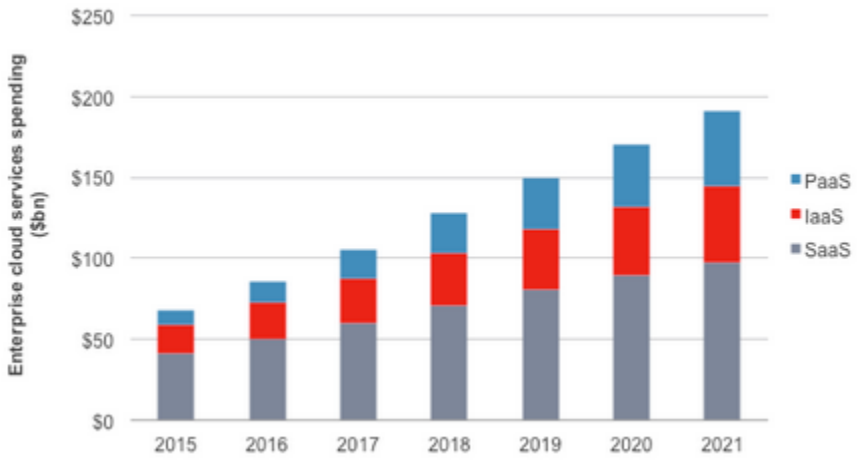

Figure 1. Enterprise Cloud services spending forecast 2016-21 ($bn)

2. Software Financial Issues: Audits And Wastes On The Rise

"*Software vendors smell money*". Thus concludes 2014 Flexera annual survey[4], observing that 65% of the interviewed companies faced at least one software license audit during the previous year. The number of companies paying more than $1 million in audit "true up" costs more than doubled in twelve months. Software audit in general are on the rise. More recently, in 2016, a BDNA[5] survey reveals the same[6]: 61% of the company panel said that they experienced at least one software license audit in the last 18 months which was close to analyst finding of 68 percent[7]. While frequency of license audits is constantly increasing, it appears that software vendors are generating a significant new revenue stream in the form of "true up" charges, paid out in addition to the original contract. They represent the penalty costs imposed by software vendors, associated for the unauthorized use of software, and have been known to impact companies with fines in the millions.

---

[4] Key Trends in Software Pricing & Licensing Survey - Software License Audits: Costs & Risks to Enterprises, conducted by Flexera Software with input from IDC's Software Pricing and Licensing Research division

[5] BDNA transforms enterprise asset data by enriching it with market context to vastly simplify integrations, accelerate business transformation and improve decision-making.

[6] BDNA Research, Does Software Asset Management really help the Software Audit Dilemma, 2016, Published August 22nd, 2016, Analyst: Cathy Won

[7] Gartner Survey Analysis: Software Audits are on the rise and your Enterprise might be Next, 2013 Published: 30 April 2013, ID: G00249225 Analyst: Jane B. Disbrow, L. Samolsky

Usually, during audit processes, the software vendor has embedded tools in their system allowing getting an account of software used licenses. The challenge is that if a company has no efficient software asset management program in place, the reliance of the data reflecting software license usage remains on the vendor's side.

While 85% of those BDNA respondents said they had an IT Asset Management (ITAM) practice in their organization, the challenge was that only 17% have ITAM tools (it includes both hardware and software asset management) in place to actually manage compliance. Ironically, 56 % of Flexera survey's panel said they are using commercial automation software to track application software usage along with license compliance. Nevertheless, 75% of companies surveyed said they remained out of compliance with software contracts last year, suggesting that current audit compliance software itself might be a waste of money. Hence, asset management and cyber security are "converging." The survey found that 73 percent of respondents monitor their systems mainly "to identify instances of unlicensed and unauthorized software on the network for cyber security purposes".

In (Table 1), we can see some remarkable figures quoted from Flexera survey about Software license audits in 2014. Flexera said that its survey reflected 489 responses, including 33 percent from enterprises with revenues of $1 billion or more. Fifty-six percent of respondents were based in the United States of America.

85 %	Percentage of organizations that are "accidental" software pirates – using more software than they have paid for
63 %	Percentage of organization audited by their software vendors in the last 18 months
34 %	Percentage of large enterprises ($3B+) audited three times or more in the last 18 months
21 %	Percentage of organizations that said they were charged $1 million or more this past year for software true ups
58 %	Percentage of enterprises that have been audited in the last year say they have been audited by Microsoft, the most frequently cited vendor doing audits
64 %	Percentage of organizations that are not using automated, commercial software to manage their software licenses
6%	Percentage of organization managing their software license manually that are satisfied with the results

Table 1. Remarkable figures quoted from Flexera survey about Software license audits in 2014

"The paradox of shelfware". When company software budgets are getting tight, it appears that many companies are wasting money on software: Flexera survey, emphasized by Gartner in 2016[8], showed that 93 % of surveyed companies are spending overwhelming amount of money on unused or underused application software, otherwise known as "shelfware". A recent InfoWorld article[9] stated that 28% of software deployed in an enterprise is unused or rarely used, and accounts for almost $7 billion of unused software worldwide.

3. The necessary emergence of Software Asset Management

The rise of shelfware and the growing number of license audits by commercial software vendors are together raising awareness of the software license risks (counterfeiting like waste). Software Asset Management (SAM) enables tracking software uses with the finest possible granularity. The aim is to constantly reconcile the real uses with the usage rights acquired from software providers in order to optimize and control the risks of non-compliance (i.e., counterfeiting).

[8] Gartner Inc. Metrics and Planning Assumptions Required to Drive Business Unit IT Strategies. April, 21st, 2016. Analyst(s): Kurt Potter | Stewart Buchanan

[9] InfoWorld, Software audits: How high tech plays hardball, April, 25th, 2016, article/3060596, by Dan Tynan.

'Software Asset Management (SAM) is all of the infrastructure and processes necessary for the effective management, control, and protection of the software assets within an organization throughout all stages of their lifecycle' (ITIL, 2011)[10]

As mentioned above, the global responsibilities of SAM are to ensure the accurate management of software assets throughout their lifecycle: from the moment it is requested, through procurement, deployment, potential recycling and finally retirement. Along with the software itself, SAM is also responsible for the license that comes with it, ensuring all users are using the software within the product use rights (PUR) and also ensuring that the organization keeps the highest standards of compliancy. Recent emergence of SAM in many companies is principally justified by two driving forces: to lower costs and to handle risks. The first is about overbuying, often seen to mitigate the risks of being out of compliance. The second is under buying: it deals with counterfeiting as soon as companies used more software than anticipated or not according to contractual clauses. This last, sometimes called "accidental piracy" is mainly due to difficulties to rightsize the software environments mainly because of the growing licensing complexity.

Moving to the cloud is a new challenge for the SAM; it represents another source of complexity and put companies in a position of using more software that they entitled to. When contracts and entitlements were based on traditional architecture models, the issue is to transfer and use the license legacy in cloud environments and slow down the incremental increase in audits for that reason. We stress the fact that editors have a right to be paid for the software their customers are consuming. The best SAM defense should be a good offense: being able to take proactive stance with a defensible audit position. SAM challenge is to eliminate the reliance on software editors for software license usage by having their own account

[10] Formally an acronym for Information Technology Infrastructure Library, ITIL is a set of detailed practices for IT service management (ITSM) that focuses on aligning IT services with the needs of business. ITIL advocates that IT services are aligned to the needs of the business and support its core processes. It provides guidance to organizations and individuals on how to use IT as a tool to facilitate business change, transformation and growth. ITIL is mapped in ISO 20000 Part 11. This recognizes the way that ITIL can be used in to meet the requirements set out for ISO 20000 certification and the interdependent nature with ITIL. This is the first such mapping that ISO (the International Organization for Standardization) has allowed to be part of their standards

of their software usage and licenses in order to minimize overspending on unused software licenses.

4. New Factors affecting Software and SAM new paradigm

In this document, we consider SAM processes in the context of emerging technologies, namely virtualization and Cloud environments. This change from traditional architectures to cloud environments, virtualized to the extreme, is still a virgin territory. Cloud environments add many degrees of complexity[11]. Among others, tracking software becomes more challenging because installation is disconnected from true physical infrastructure. Altogether, the complexity of software lifecycle management, the multiplication of actors in this cycle and the lack of efficient tools, lead to an understandable disconnection between software usages, associated hardware and the related licensing model. Also, because cloud environments tend to automate software lifecycle management, SAM processes are expected to be automated as well. On the contrary, automation is currently circumscribed to asset management in traditional architecture.

Going further, in cloud environments, SAM is not only assets management, but also service management, which must be done in real time taking into account the fast rhythm of changes: services are provisioned, configured, reconfigured and decommissioned in a matter of minutes. Compliance risks are increased by the ease and speed of provisioning, which can bypass traditional centralized processes. In such conditions, SAM controls are difficult to implement. The idea that will be developed is that turning to the Cloud is not changing the object of SAM, but altering how SAM processes should be designed.

Some techno-economic drivers are converging to create a paradigm change in the design and operation of future telecommunications networks and services. These drivers encompass progress in Information Technologies (IT), pervasive diffusion of ultra-broadband access, commoditization and falling costs of hardware, and the maturity of virtualization techniques. Network Function Virtualization (NFV) is a concept pushed by the industry to virtualize network equipment using generic-built hardware platforms, in order to reduce costs and increase network

[11] M. McRoberts, Software Licensing in the Cloud Age : soling the Impact of Cloud Computing on Software Licensing Models, The International Journal of Soft Computing and Software Engineering [JSCSE], Vol. 3, No. 3, , San Francisco State University, CA, U.S.A., March 2013Doi: 10.7321/jscse.v3.n3.60e-ISSN: 2251-7545

operation and performance efficiency/agility. The NFV concept separates network functions from the hardware they run on using virtual hardware abstraction, and attempts to virtualize entire classes of network node functions into building blocks that may be connected or chained together to create communication services. Alike, "Softwarization" is an overall techno-economic transformation impacting the design, implementation, deployment and operations of infrastructures, deeply integrating network nodes and IT systems. For both network functions and services, flexibility and agility of software is highlighted. This transformation enables new architectural models along with an automation of operational processes. All these considerations force us to question a new dimension of network management: as software becomes omnipresent, we assume that software license's management in real-time and on large-scale cloud environment will sophisticate Virtualized Network Function (VNF, or Network Software) on-boarding processes. Network virtualization and softwarization lead to a disruption in terms of software licensing business model; thereby, we develop here the necessity to adopt existing and relevant software license optimization IT process. We do believe that this experience and expertise acquired from IT will facilitate this NFV turn. In other words Software Asset Management (SAM) should play a major role in defining best practices the network industry could follow.

5. CONTRIBUTIONS AND FOLLOWINGS

The following contributions are spread through three years collaboration with Orange SA, an international telecommunication company. Thereby, the industrial input address the possibility for Orange to propose new licensing model designed for it cloud and virtualized network architectures through the development of a prototype "cSAM": a solution to analyze the real and dynamic usages of software resources in the cloud. The aim is to ensure compliance, to determine real costs for users, to optimize the deployment of licenses based on predefined and adjustable scenario and finally to strengthen Orange position facing editors including the creation of a software user open-community. cSAM value-added is to integrate cloud dynamicity issues, to be flexible and multi-domains, to integrate new and complex metrics (business models) and to propose innovative simulation functions to allow better uses and deployment controls.

Thereby, we propose (i) a SAM maturity scale, (ii) an architecture for SAM in the cloud, (iii) the related SAM management workflow, (iv) some major implementation choices and (v) their evaluation; furthermore we question (vi) the

emerging contractual relation trends between service providers and software editors; (vii) we argue that SAM is necessary in NFV environments and (viii) we propose a SAM prerequisite approach for NFV environments.

The remaining of this dissertation is organized as follows: we propose in section 2 an evaluation of the academic state of the art, an evaluation grid and practical application on the SAM tools proposed by the market. Section 3 proposes requirements for accurate SAM identification, management flow in cloud environment and workaround propositions for its implementation. Section 4 proposes a SAM model for the cloud based on SAM processes control loop and lifecycle identification and a database model for SAM loop. Section 5 proposes a qualitative evaluation of our works based on model assessments for two use-cases: on a PaaS layer and on a Network Function Virtualization (NFV) platform. We conclude in Section 6.

Chapter 2

II. STATE OF THE ART

State of the art 37

1. Academic State of The Art 38

A. Theoretical works about SAM and its implementation limits 38

01. SAM organizational implementation's limits 40

02. SAM technical implementation's limits 40

B. Nowadays context justifies emergence of SAM in the literature 41

C. Synthesis: SAM maturity scale 42

2. Industrial State of The Art 44

A. Requirements 44

B. Evaluation grid 44

C. Choice of SAM Market tools 49

D. Results of the evaluation 49

01. Aspera Software Smart Track 49

02. Snow License Manager 8 53

03. Flexera FlexNet Manager 56

04. Spider 59

05. Eracent 61

06. HP Asset Manager 63

07. BMC Remedy 65

08. GLPI – OCSng 67

E. Synthesis on Industrial State of the Art 69

3. Synthesis 70

We propose to classify this chapter in (1) an academic state of the art and (2) an industrial state of the art mostly based on market tool analysis. (3) We propose as a synthesis a SAM maturity scale and analysis of SAM complexity factors brought by cloud architectures.

1. Academic State of The Art

We can underline the low amount of academic publications dealing with Software Asset Management comparing to the last decade proliferation of industrial white papers and analyst's recommendations. Yet, in the last few years, the slowly growing amount of patents related with license management solutions point out the receptiveness and permeability of this industrial concern about SAM. Moreover, more and more every day, Software is considered as a consumable no more only like an asset; resource consumption especially in virtualization context is a booming concern in the scientific literature.

1.1. Theoretical works about SAM and its implementation limits

The idea spread that Software asset management is crucial to the success of any IT organization. When a company has a comprehensive and efficient license management program in place, it reduces costs and ensures that the organization remains in compliance.

(N.F. Holsing, 1999) [2] proposed a software asset probation model and identification of five problem areas which drive the need for software management: ethical (intellectual property rights' respect), legal (counterfeiting), technical (monitoring), managerial and economic issues (true-up costs), when identified, lead implementation of SAM within an organization, from different parties' viewpoints: end-user, employer and software editor. The authors developed the idea that the main goal of SAM is to ensure the software license compliance through employee education which provides the groundwork for legal and cost effective uses of software.

(M. Ben-Menachem, 2004) [3] introduced the "paradigm of change" based on methods, tools and procedure for an accurate overall IT inventory management. For them, one of the most significant failures of IT is the lack of systems to gather, support, and supply information for managing software items. Most IT

professionals, if they consider software management, think in terms of version or configuration control license and patch management. Version control systems and software configuration management systems aim to manage versions of individual software objects with support for linking into sets for release purposes. This has nothing to do with addressing the issues of controlling large amounts of geographically disbursed software, executing on different kinds of systems, maintained by hundreds to thousands of programmers. For the authors, software systems are the only major organizational asset with no real support for managing them based on information technology. An appropriate IT inventory management facility is the cornerstone of an integrated set of technologies ("Paradigm of change") designed to address constantly changing technologies and business processes. Thereby, they underlined that investment in creation and maintenance of dedicated software inventory is sine qua non prerequisite to proper long-term software asset management. (M. Ben-Menachem, 2005) **Erreur ! Source du renvoi introuvable.**[4] defined in addition a methodology software control by importance and exception.

(M.McCarthy, 2011) [5] proposed a solution in four points to combined IT, processes and business in SAM perspectives:

- Discover Software Assets
  - Agents scan/discover distributed software license assets
  - Software licenses are linked to employee & workstation
  - Scan data populates asset database as discovered inventory
  - Provides base line for audit compliance reporting
- Reconcile Purchased Assets
  - Reconcile software procurement inventory
  - Life cycle management of purchased 3rd party software licenses
  - Sustained asset reconciliation and compliance
  - Leverage global purchasing power
- Implement Contract Management
  - Compliance with License Terms & Condition
  - Enables reuse of licenses through off-loading (attrition, allocation, entitlement)
  - Enables governance and process automation
- Produce Business Intelligence Reporting
  - Audit readiness and compliance
  - Analyze, track, & forecast global IT software spend
  - Executive and management reports proactively target audit compliance risks

Result of the experience showed that the solution's out-of-the-box capabilities, comprehensive analytics, workflow automation and business controls features immediately improved time-to-value, helping their organization realize more than US$5 million in savings in the first year of deployment. It represents the foundations of the SAM which was defined officially the same year by ITIL showing the industry/literature concordance.

a. SAM organizational implementation's limits

In the 2000s, when industrial concerns emerged about the necessity to monitor software usage, the literature started addressing this topic by way of limits in Software Asset Management. In order to explain the difficulty of setting up SAM processes in medium-large organizations.

First limitation is about vague software lifecycle. Software is an intangible asset, distributed as equally immaterial license, negotiated by buyers on the base of contracts approved by layers, for dedicated purpose of a team usage, installed by exploitation teams. (M.Sharifi, 2009) [6] explained that organizations are under the pressures of managing software systems which are bigger and more complex than those from past years, but also need to meet increasing demands for higher quality to meet organization's objectives. One important problem is that most organizations do not know how much software is running in their organizations. The problem is increased by the fact that software is not visible and has a tendency to live forever.

Literature also describes how vague software lifecycle leads to hazy responsibilities. This is mainly explained by the lack of communication between lifecycle stakeholders. (M.Benachem, 2008) [7] showed that IT department's inability to document and justify their expenses prevent CFOs and CEOs SAM initiatives. He underlined basic issue of information transferability and lack of interdepartmental data sharing.

b. SAM technical implementation's limits

The second main limitation is about tracking software: a common mistake is to underestimate the process of identifying software. The comparison between contractual, installation and usage data is laborious due to their heterogeneity. The lack of efficient tool (called "Excel sheet management") was pointed by many authors as a main challenge for the SAM: (Klint and Verhoel, 2002) [8] shown that lack of inventory information blind organizations in terms of total IT spends.

(Ben Menachem, 2004) pointed the fact that major organizations have very primitive or out of date assets inventory or central repository.

(McCarthy and Herger, 2010) [9] identified that the lack of tools to measure and monitor usage and availability of software licenses make difficult to measure software asset uses, creating compliance issues.

1.2. NOWADAYS CONTEXT JUSTIFIES EMERGENCE OF SAM IN THE LITERATURE

(A.Manzalini, 2015)[10] stated that Network Function Virtualization (NFV) principles are going to impact not only the evolution of current networks, but also the services and applications platforms. He argued that, in this evolution, the border between the networks and the Cloud-Edge Computing platforms will gradually disappear. As well the distinction between the networks and the future "terminals" (i.e., devices, smart objects, drones, and robot) will blur.

(C.Matsumoto, 2014)[11] The promise of NFV is to move network functions out of specialized appliances onto off-the-shelf servers. The objective is both to save money and to gain regarding the time factor. The normal process of installing new gear for new services can take weeks. (R. Jones, 2016)[12] promising agility and flexibility, some network software vendors say NFV can shrink that process down to minutes. Many challenges are involved in deploying and operating a cloud-based NFV platform. (L.M.Contreras, 2015)[13] Virtualization and dynamic "on-demand" services bring new challenges for traditional network ecosystems which were used to have license keys to enforce entitlement. In NFV or other virtualized environments, virtualization facilitates "copy/ distribute/run" application and software. VNFs have a passing lifecycle, are not typically locked to a physical host. Having available licenses key at the right time and place drives administrative costs for a global distributed cloud system, such as a NFV infrastructure (M. Adler, 2014)[14].

Cloud computing is revolutionizing the way organizations pay for and use their IT resources.(M.McRoberts, 2013)[15] has shown that while cloud computing has the potential to simplify the licensing and use of software, it has, in fact, only added to the problem. For commercial software vendors to successfully move into the cloud age, they must work as a group with cloud providers to standardize licensing in the cloud. Standards-developing organizations should govern the activity. A successful solution must address legal and financial concerns, as well a technical aspects of software licensing in the cloud.

As well, regarding NFV, software vendors have relationships with service providers, who, in the long run, need to integrate with a vendor NFV platform. By convention, VNF vendors have been selling their VNF products directly to service providers. For the latter, there is a need for homemade or third-party integration and bundling of VNF products together to reduce operational expenses and/or engineering expenses. For some it would be advantageous to have a pluggable

framework for a cloud-based NFV system that allowed integration of VNF products to provide a diverse catalog of VNF services in an integrated manner. As an example (R. Jones, 2016)proposes a dynamic licensing method, implemented in an integrated system, including a third-party application; an exchange of private/public keys transiting through the integrated system validates the validity of the application's license key, determining whether to run the application.

1.3. SYNTHESIS: SAM MATURITY SCALE

SAM enables tracking software uses with the finest possible granularity. The aim is to constantly reconcile the real uses with the usage rights acquired from software providers in order to optimize and control the risks of non-compliance (i.e., counterfeiting). Cloud environments add many degrees of complexity. Among others, tracking software becomes more challenging because installation is disconnected from true physical infrastructure. Altogether, the complexity of software lifecycle management, the multiplication of actors in this cycle and the lack of efficient tools, lead to an understandable disconnection between software usages, associated hardware and the related licensing model. Also, because cloud environments tend to automate software lifecycle management, SAM processes are expected to be integrated and automated as well. On the contrary, automation is currently circumscribed to asset management in traditional architecture. Going further, in virtualized environments, SAM is not only assets management, but also service management, which must be done in real time taking into account the fast rhythm of changes: services are provisioned, configured, reconfigured and terminated, retired in a matter of minutes. Compliance risks are increased by the ease and speed of provisioning, which can bypass traditional centralized processes. In such conditions, SAM controls are challenging to implement.

Based on the currents, we propose in Fig. 2, our evaluation of SAM maturity on two axes. This scale allows focusing on SAM processes adding a "cloud ready" dimension. Four levels can be defined on a vertical axis about SAM maturity. Each level has to be supported by tools to perform efficient actions.

Figure 2 - Proposition for a SAM maturity scale

The first is entitled VISIBILITY: it consists in a precise resource and asset identification. In other words it consist in recognizing each device, with its physical features; to identify lifecycle of virtual machines and resources allocated to it and to discover all software which are installed on any physical or virtual devices.

The second level: IDENTIFICATION consists in translating all software installation in terms of related licenses and products user rights. It can be identifying a product as a trial version or circumscribed to a particular scope; diagnose that it belong to a software suite or that it is an option which use is conditioned by the use of the basic product. it is also identification of all usages, to be able to discover and translate in terms of usage rights, all possible access to a software.

The third level, RISK MANAGEMENT consists in reconciliation of data from contracts (which specifies product usage rights), from installations (technical view) and from real usages. Mainly, the aim is to prevent two different risks: the first one is a legal one, piracy: you are using software without license or with wrong way of licensing (accidental piracy, often due to the complexity of licensing models today). The second is a financial risk, over-deployment: you are not using licensed software, or your license is covering more usage rights than needed.

The fourth level is OPTIMIZATION: when you have an accurate view of your usages and assets, you have to identify all possibility to improve both your license spends and architecture of your installations.

The fact is that all this four levels do not have the same maturity. A lot of tools are really efficient in terms of discovery of assets on equipped resources. More problematic is the second level, especially because matching between information from contracts, usages and technical view from first level is, at least, not easy. In this situation, despite numerous tools of risk management, treatments are approximate and optimization cannot be automatized.

2. Industrial State of The Art

2.1. Requirements

As a SAM we need to know, in real time, the status of the license stocks: therefore as close as possible to the real time, we need to confront the software use with the license stock according to a measure of consumption previously defined (called metric). It implies that we can precisely identify the allocated resource chain (through each layer of virtualization) and obtain the features needed to measure usage and lifecycle software specifications on machines. We have to integrate constraints imposed by the nature of the product or its uses (i.e. options, technology stack: a combination of software products and programming languages used to create an application). These constraints may involve links between products. We must identify situation of multiple access and translate it in terms of use (Bring your own device (BYOD), multiplexing, multidevice …). We must be able to anticipate organization needs as close as possible to real time: to create and realize different scenarios based on the evolution of the consumption (including automated process of adjustment); to create cost models for any measure of use and identify the most suitable scenario of consumption for the customer's billing.

Regularly, the tool must be able to prove its relevance especially with reliable, accurate and auditable historic of established uses. We must monitor and follow update of any product to detect and monitor related services (i.e., case of maintenance). All information collected and analyzed should help to propose legally, financially and technically acceptable models.

2.2. Evaluation grid

Software Asset Management processes like decision making about purchase, management or elimination of software, have to be support by tools (for each four levels described above). (M. Thompson, 2017) [16] comparing existing SAM tools is challenging for the following reasons (among others):

It is easy to notice the exuberant marketing made by publishers about features that appears similar between existing tools and the lack of model to classify them. The scope is also absolutely not defined between traditional architecture and cloud environment, as if the way to manage software assets in both environment was similar. The proliferation of tools is also due to multiplication of actions to manage (as explained in the four levels scale above). For example: management tools often perform discovery activities and inventory, but they rarely gather sufficient details on the software inventory to allow decision making, or compare inventory data to the product use rights acquired in the contracts.

Based on the SAM maturity scale presented in synthesis (1.C), we can propose a tool classification grid to evaluate performance of common tools proposed by the market (open-source & proprietary software). As the SAM maturity scale can be read on two axis (vertical for activities, horizontal for traditional/cloud architecture), this grid should be read on the two same axis. It is organized in 6 + 1 items (Tab.2)

VISIBILITY
IDENTIFICATION
RISK MANAGEMENT
OPTIMIZATION
DECISION MAKING
CONTINIOUS IMPROVEMENT
(COST)

Table 2 - SAM Maturity Items

Below, we provide a high level summary of the six major areas (in annexes, the full grid used for the evaluation).

- **Visibility**: We want to check if the technology can track and manage infrastructure up to the existence and usage of virtual platforms, virtual operating systems or web based applications (each virtualization layers). If the tool can tell where the virtual machines are and how they relate it to users, locations and physical machines?
    - Items:

        - Identifying (and maintain list of) all assets
        - Scope of identification
        - Communication with assets
        - Take organization into account
        - Level of virtualization
        - Dynamic partitioning
        - Environment
    - Our observations: Cloud and mobile discovery starts to emerge. But accuracy of data is still a weak point.

- **Identification**: We want to check that the tool can recognize software titles from raw technical data; identify all usages in the finest granularity (disconnection from contract's metric); identify the product use rights for all software, manage entitlement statements from software publishers and integrate with procurement systems. Manage complex license types and bespoke negotiated clauses.
    - Items:
        - Recognition of software license needs
        - Inter-software products links
        - Additional elements rise
        - Prioritization of products
        - Identification of software uses
        - Which level of automatization
        - How to reconcile product and rights
        - How to reconcile real usages and metric
        - Database access
        - Contracts management
    - Our observations: Identification of a licensable status is a core competence for modern tools, but still fragile because conditioned by fragile processes of recognition. Accuracy of the data is a critical issue, not solved for the moment, especially in Cloud environment.

- **Risk Management**: What intelligence is provided to software asset manager to assess that they are in compliance position, giving possibility to re-negotiate contracts and remove risk?
    - Items :
        - Compliance verification
        - Confidentiality of data
        - Auto-Allocation of license

        - Alerts
        - Safety and permanence of data
    - Our observation: Compliance statements are promised by major part of tools. Of course, result is conditioned by accuracy of data brought by inventory process

- **Optimization**: Reporting on what applications are not being used, identifying opportunities to renegotiate metric more fitted to real usages. Identifying suite or functional overlap, suggesting open-sources or cheaper alternatives make smarter decisions on maintenance or renewals subscription, benchmarking spends and usages.
    - Items:
        - Usage measurement and interface with inventory
        - Corrective action
        - Maladjusted usage detection
        - Portfolios consolidation
        - Maintenance contract optimization
        - Architecture optimization
    - Our observations: Software usage is common among those SAM that offer inventory but software optimization is under used. There is significant further opportunity to optimize using simulations features.

- **Decision making**: Being proactive stakeholder in all actions and processes which can have impact(s) on software lifecycle. Service request automation, catalogues, automated processes, ITSM lifecycle integrations, scenario modelling, advanced reporting, internal markets.
    - Items:
        - Scenario studies
        - Helping IT to make decision
        - Helping Buyers to make decision
        - Help-Desk leverage
        - Helping Audit process (User/device advisor)
    - Our observations: Anticipating and helping SAM to react on changes should be enhanced by tools. Still need to be implemented. Some theoretical works on it for traditional SAM, nothing about cloud environments.

- **Continuous Improvement**: How the system can enrich the all processes and how the system can be easily enriched? What is the level of technical expertise needed to access this solution? - What will be TCO of this solution?
    - Items:
        - Processes reliability
        - Concepts modularity
        - Initial cost
        - Modification/adaptation costs
        - Access to support
        - Technical debt
    - Our observations: As more logical is the deployment of assets offered by virtualized environment as better are be the possibility to find optimization both in technical architecture and in license spends.

Depending the context, the weight of these items in the evaluation of SAM tool's efficiency may change. Indeed, VISIBILITY, in traditional architecture is no more an issue, because a lot of discovery tools are quite efficient in devices detections. In cloud environment, it starts to be more difficult to have precise and REAL TIME view of all resources. Moreover, disconnection between hardware and software in the cloud makes more difficult this recognition and link between assets: IDENTIFICATION remains crucial point, especially in terms of usages.

From this finding, with focus on cloud perspective, we can attribute like shown in Fig.3, the following weight to each ITEM (cost excluded):

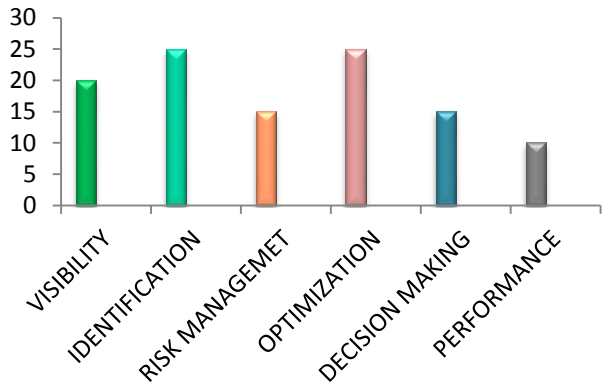

Figure 3 - Study's criteria weight

2.3. CHOICE OF SAM MARKET TOOLS

Our aim is to provide an independent review and comparison of the market leaders, identify key competitive differentiators between tools and confront what the market is heading with our requirements presented in 2.a. The evaluation is a broad competitive comparison of market leading SAM tools for large companies. The choice of the selected tools was based on several criteria, including the opportunity to test it (i.e. for Aspera Smart Track and Flexera Flexnet Manager) or to benefit from detailed feedback and own experimentations (i.e., internal feedbacks in Orange for GLPI/OCS, BMC Remedy, Snow License Manager and HP Asset Manager). We choose major SAM editor's products and included a couple of open-source products (GLPI – OCSng) whose user communities are the most active (M.Thompson, 2015)[17].

- Aspera Smart track [18]
- Snow License Manager [19]
- Flexera Flexnet Manager [20]
- Spider Brainware [21]
- Eracent [22]
- HP Asset Manager [23]
- BMC Remedy [24]
- GLPI – OCSng [25]

2.4. RESULTS OF THE EVALUATION

The figures 4 up to 19 synthetize the results of evaluation. For each tool, a first chart represents marks based on the seven criteria described above; a second chart gives a mark to specific items for each of the criteria.

a. Aspera Software Smart Track

One of the SAM market's leader Aspera's offer want to be defined like "optimize the right products to deploy, and deploy in the right way". SmartTrack offer an intuitive and user-friendly web based console interface.

Oriented on license management: contrary to its competitors who develop in addition inventory/security/delivery tools or modules, Aspera does not provide built-in discovery and inventory solution. (To overcome gap in inventory and discovery coverage, Aspera works with the likes of iQuate and Raynet). Weak point can be that if SmartTrack facilitates the license management within other tools

such as service desk, it does not provide app possibility to implement process leading to creation of a single tool within entity.

Aspera's strengths lie in compliance and optimization: Aspera addresses software compliance very well. We can underline efficiency and cleverness of the catalog's data records automatically transferred to SmartTrack and seamlessly linked to the metric engine algorithms. Aspera is transparent with calculations; you can clearly see both the license metrics workings and whether gaps exist in building an accurate license position. Future versions of the interface will include the ability to build custom metrics into SmartTrack dashboards. However, Aspera could improve emphasis on the data quality and import regarding inventory sources: i.e. the tool highlights that missing data can generate gaps in recognition for license management but on higher perspective does not show that the data you imported is exhaustive and covers your entire estate by comparing and confronting imported inventory sources. Aspera is designed for ongoing cost optimization as well as point in time compliance; SmartTrack continually lists optimization opportunities including comparing price points against the customers average acquisition price to identify unnecessarily high unit costs. SmartTrack provides a guide price based on previous procurement record entries, reseller pricing or vendor price lists. SmartTrack also helps clients to fully exploit their product use rights and making best use of their existing entitlement.

Interesting simulation beginnings: SAM can operate SmartTrack to forecast the costs of some different architectures (mainly CPU changes), renewal or metric considerations. The simulation allows building clusters, incorporating existing licenses and historical purchases. Interesting visualization gives a topology of datacenter environments and visual virtual relationships.

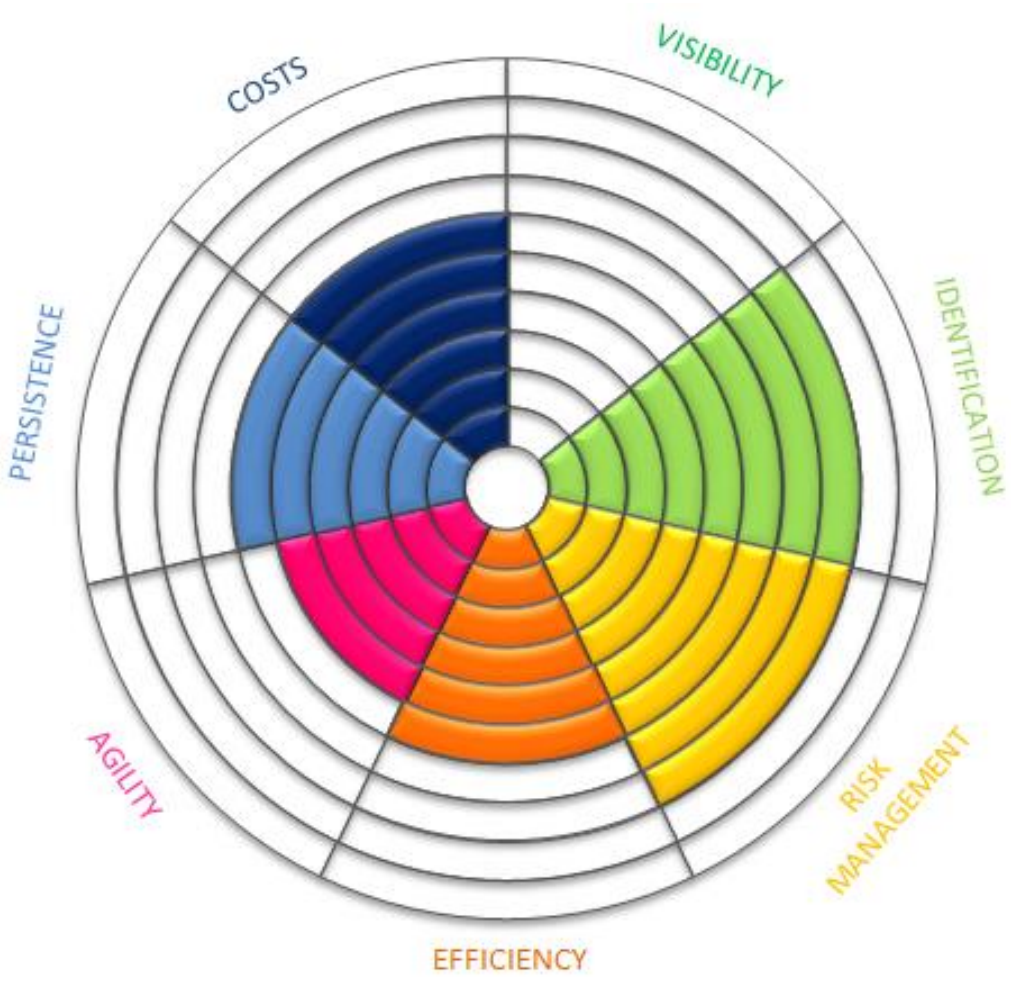

Figure 4 - Aspera Evaluation Summary

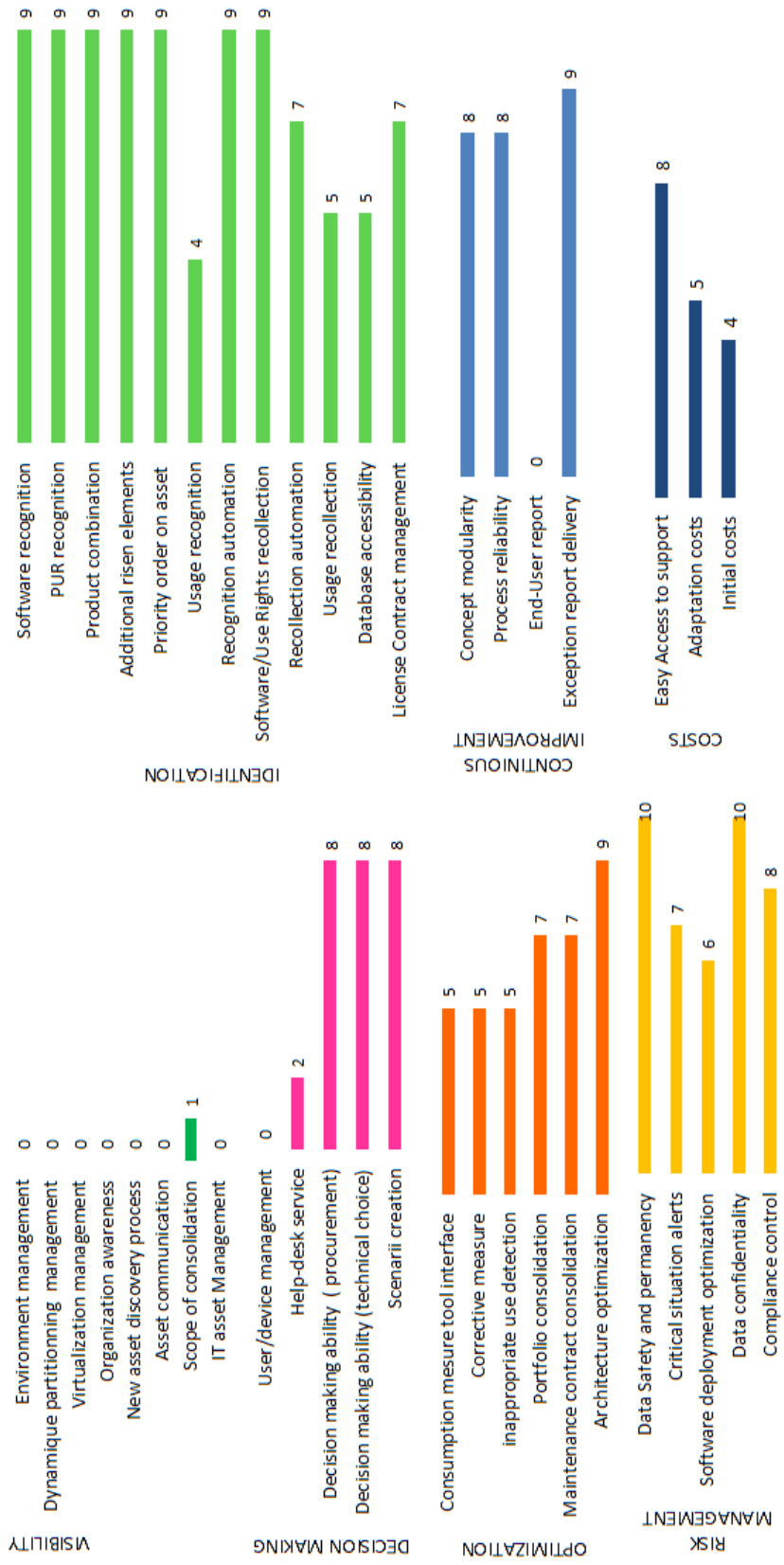

Figure 5 - Aspera Evaluation detailed view

b. Snow License Manager 8

A flexible solution from desktop to datacenter : Snow proposes a competitive solution to measure consumption of software from mobile/tablet, desktop, virtual machine, hypervisor, to cluster and even data center. Collecting data is getting easier especially using the Snow inventory client, which supplement inventory sources with the data necessary to measure consumption. The License Manager has 18 out-of-the-box connectors to 3rd party inventory sources or an XML based connector to connect to anything else. Snow is equally soft in handling business data (procurements), which can be automated in the same way as the input of technical configuration data. We can underline the ease of use and simplicity of Snow License Management to handle complex objects and show easy-to-understand results. Snow is clearly oriented on fast cycles and agile deployments, less than on customizable route of software tools. We can commend Snow's transparency and ability to show at the same time data and its origin/provenance thereby always being audit-ready or finding negotiation leverages (latest version).

Pioneer in Software Recognition: Its Software Recognition Service recognizes commercial software in a couple of days. Snow started to enrich this process via direct relationships with software editors to ensure more accurate and relevant recognition (i.e. Autodesk and Red Hat).

From interesting strategic functions to weak strategic planning: Snow License Manager provides interesting views oriented on consumption and financial optimization and both can be put in perspective by a well-managed historic. Snow matches the overall trend for SAM to move from an administrative function to strategic, while Snow's competitors propose stronger functions of scenario modeling and strategic planning.

Valuable Snow Automation Platform: It gives organizations the ability to automate and integrate a diverse range of processes that contribute to the overall effectiveness of SAM. From facilitating the bi-directional exchange of information between the Snow SAM platform and other systems, to automating the process for software requests and re-harvesting, the Snow Automation Platform is the key to mapping the inherent capabilities in Snow License Manager into the organization's individual SAM processes.

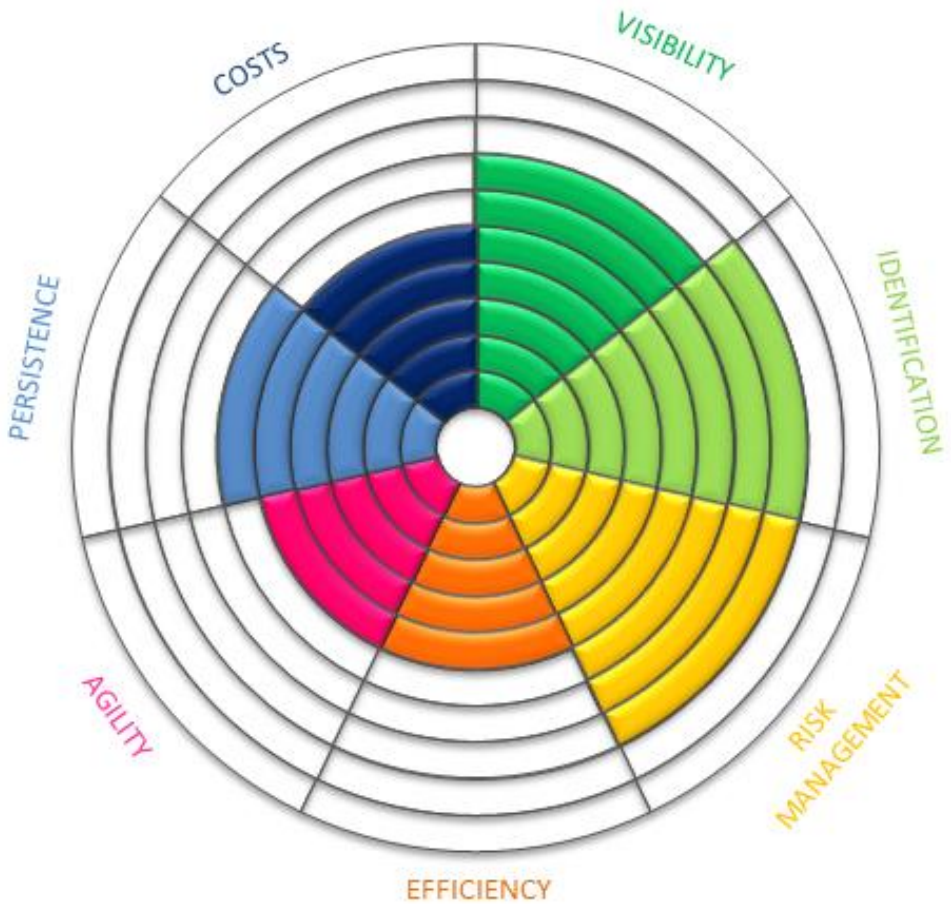

Figure 6 - Snow Software Evaluation Summary

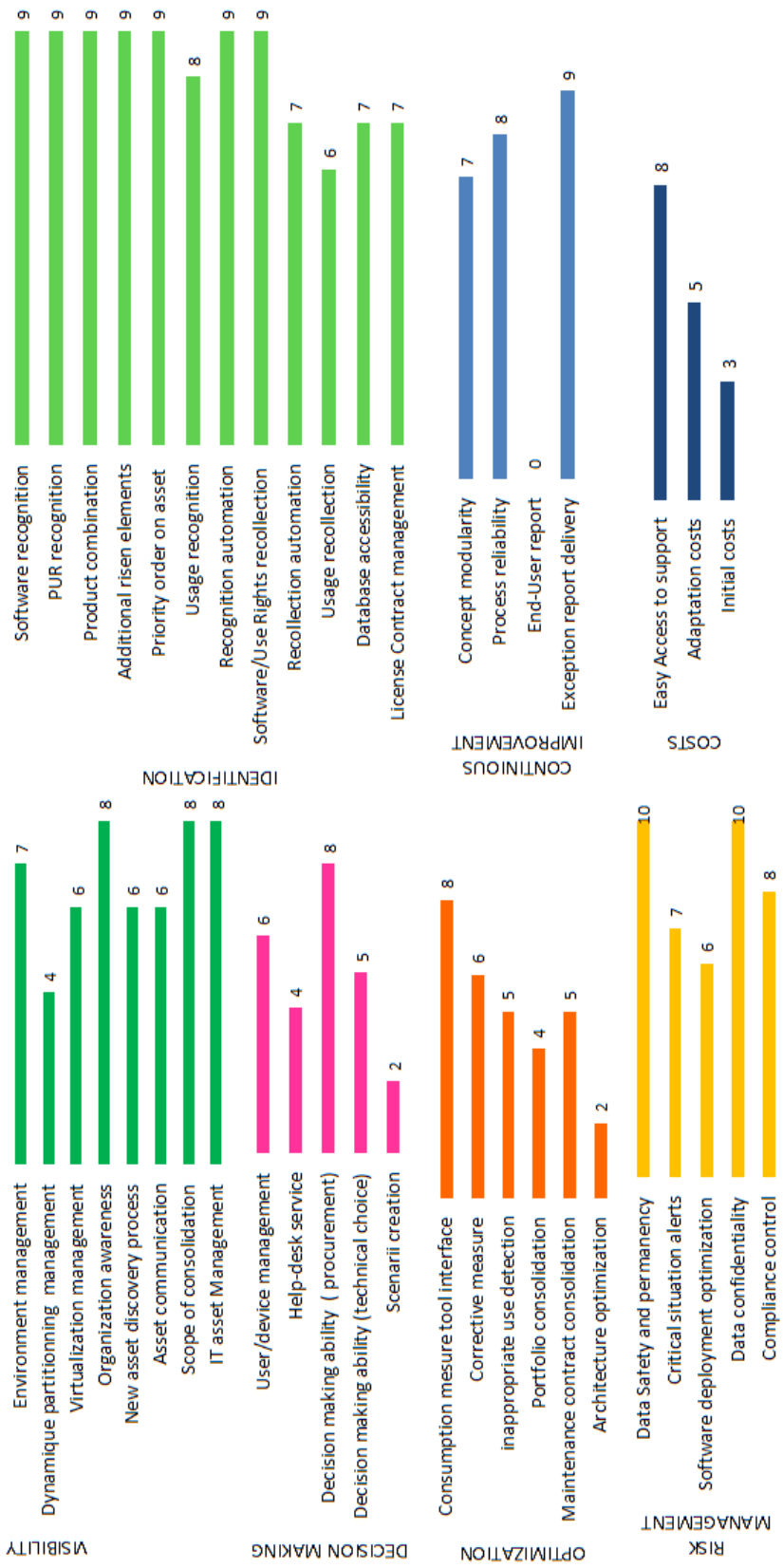

Figure 7 - Snow Software Evaluation detailed view

c. Flexera FlexNet Manager

A durable competitive solution in the SAM market: With FlexNet Manager, Flexera proposes a challenging solution for Software Asset Management and optimization. We can underline a solid dynamic license management, interesting financial optimization features and ongoing strategic possibilities. Once implemented with an appropriate SAM team and resources, FlexNet Manager is an efficient visibility booth of software risk, optimizing spend and planning for the future.

User oriented: Comparing with Flexera's competitors, the user interface and the quality of dashboards are less attractive and user-friendly but we appreciate the ability for software responsibilities to be delegated to end user customers via their own login (in App Portal). Flexera's Application Portal product allows users to request a wide range of authorized applications, including SaaS apps as well as desktop and mobile apps. Some of Flexera's competitors also offer single sign on solutions to automate provision of SaaS from within an app store. Flexera's proclaimed goal is helping customers with the large complex environments, contracts and IT challenges such as virtualization, cloud and BYOD. Flexera includes management of Amazon Web Services cloud infrastructure costs and utilization (via FlexNet Manager for Cloud Infrastructure), as well as further development of their App Portal enterprise app store offering. Flexera's competitors are getting closer on the Enterprise SAM space and some of Flexera's competitive differentiators, such as Oracle verification, are based on software publisher verification rather than genuine technological innovation. Feedback from Flexera's customers on Tools Advisor suggests upgrades and enhancements can be labour intensive.

Good performances on software recognition: Flexera's application recognition library already contains 180,000 software titles, while its Product Use Rights libraries include license characteristics such as processor point's tables, upgrade and downgrade rights, mobility rights and so on – adding vulnerabilities into the mix in the longer term can only add value to their core

A foray to note into the security domain: We can underline that Flexera acquired and included in offer Secunia, which provides visibility and risk assessments of software vulnerabilities on end points. It is strategic dissimilarity for Flexera; the addition of Secunia completes the software management trilogy of packaging, asset management and security around applications. On paper Secunia is a competitive differentiator but it may also prove to be a distraction in comparison to Flexera's key SAM competitors who focus only on SAM.

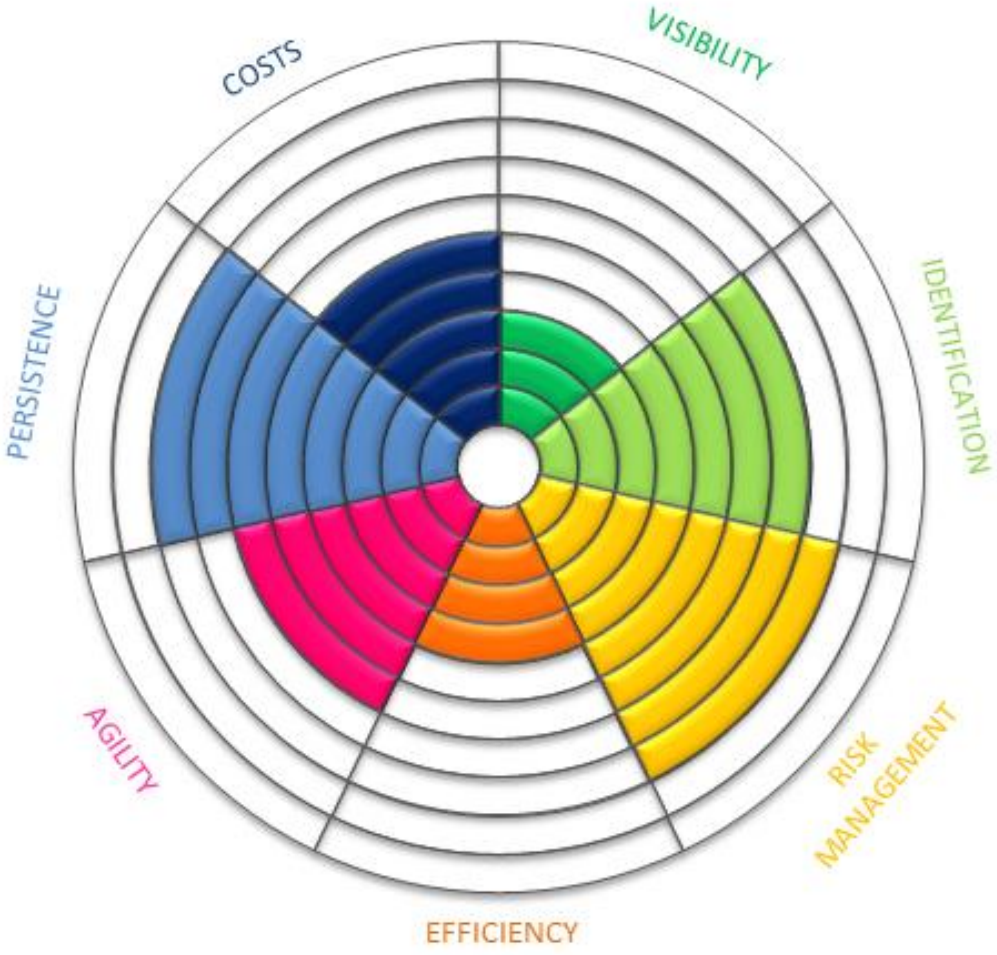

Figure 8 - Flexera Evaluation summary

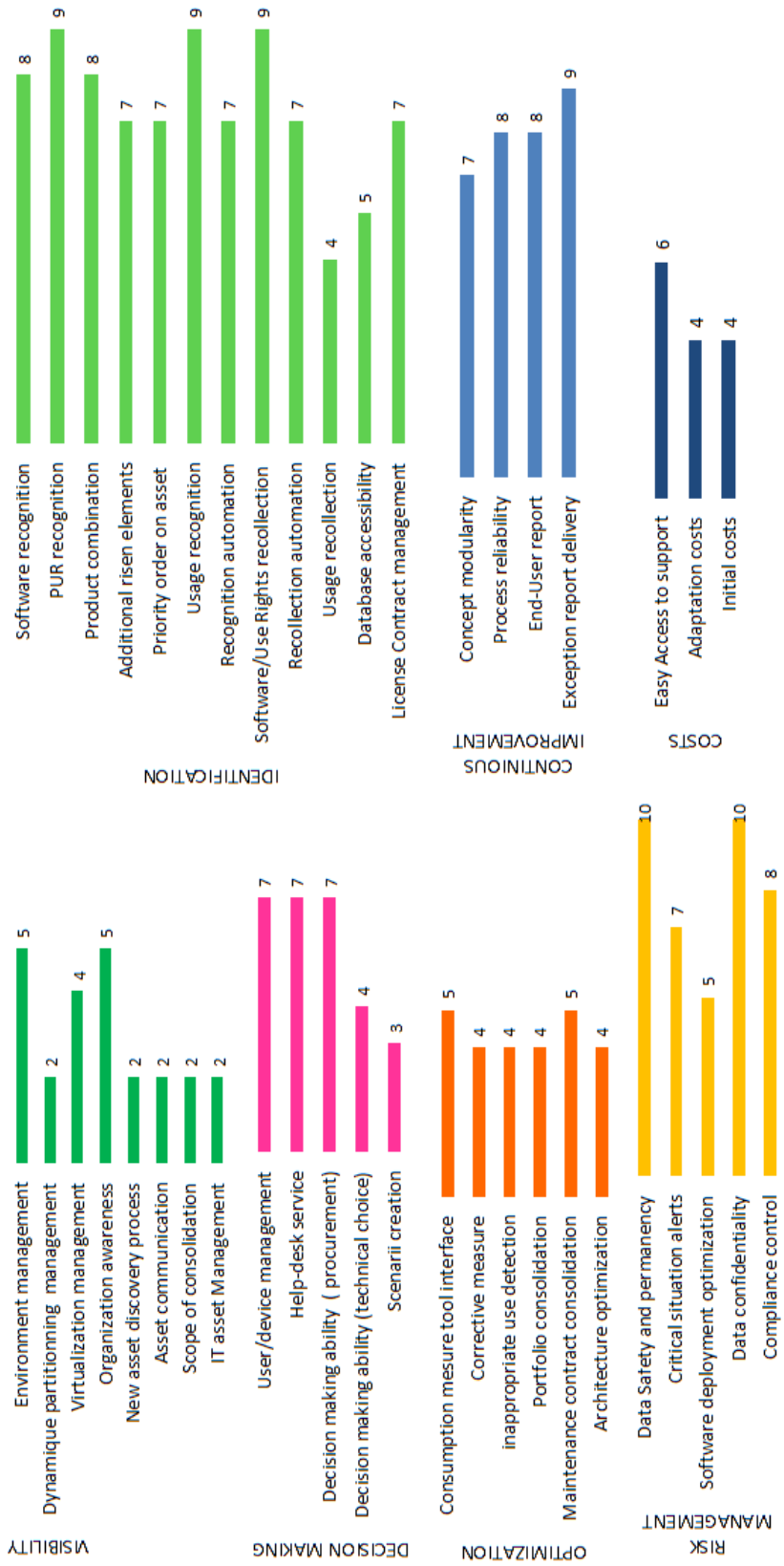

Figure 9. Flexera Evaluation Detailed view

d. Spider

As strengths, we underline the Spider Brainware ability to bring in multiple data sources, combined with great flexibility; uncluttered interface individual configurations possible; it has a very good license and asset management know-how. Yet, it is more an overall IT Asset Management than a dedicated tool only focused on SAM. We regret lack of an internal workflow engine which leads to weak search in contract and core data (Core is becoming increasingly important for user licenses / cloud offerings). The compliance view might be too confusing and reporting are not easy to get (mainly because of the interface).We also underline weaknesses of the product catalog.

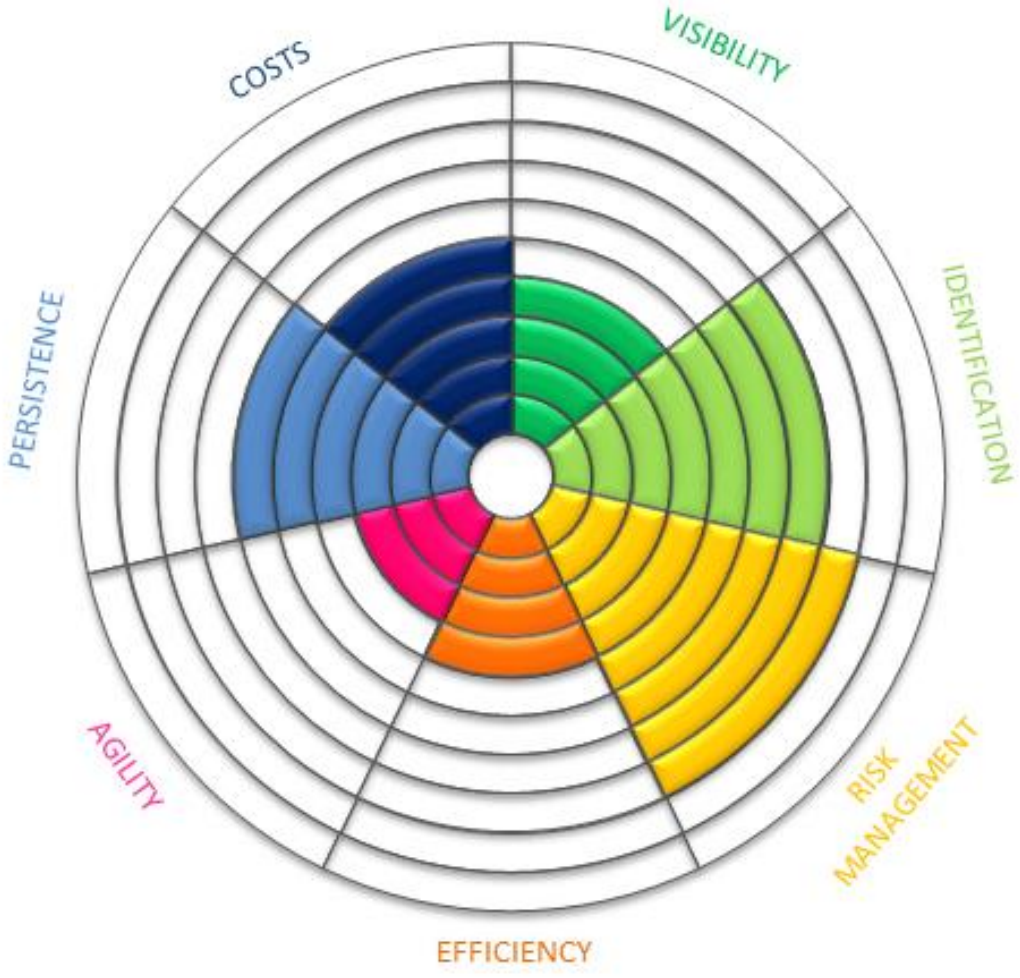

Figure 10 - Spider Evaluation Summary

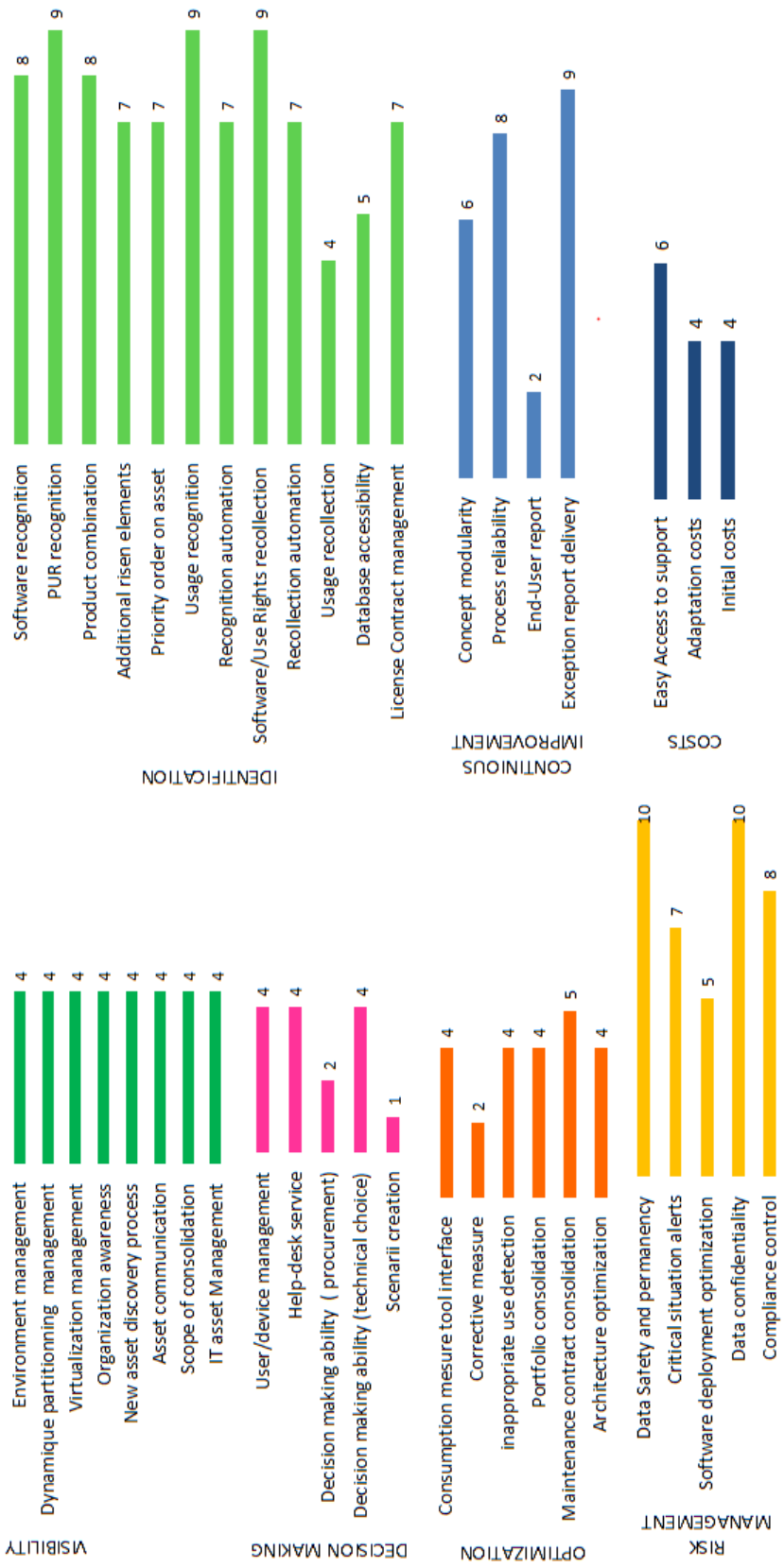

Figure 11 - Spider Evaluation Detailed view

e. Eracent

Eracent has a very comprehensive agent for discovery of both hardware and software on a daily basis, as well as during software vendor audits. A special good point for the robust Lifecycle Management capabilities of Eracent. As weaknesses we first point that: Eracent continues to enhance and improve the UI for the Software License Entitlement and Reconciliation portion of their product. The recently added CLR (Continuous License Reconciliation) feature provides detailed software license reconciliation data as well as high level graphical summaries. We look forward to the future enhancements that Eracent has on their roadmap. For the second we regret a lack of documentation.

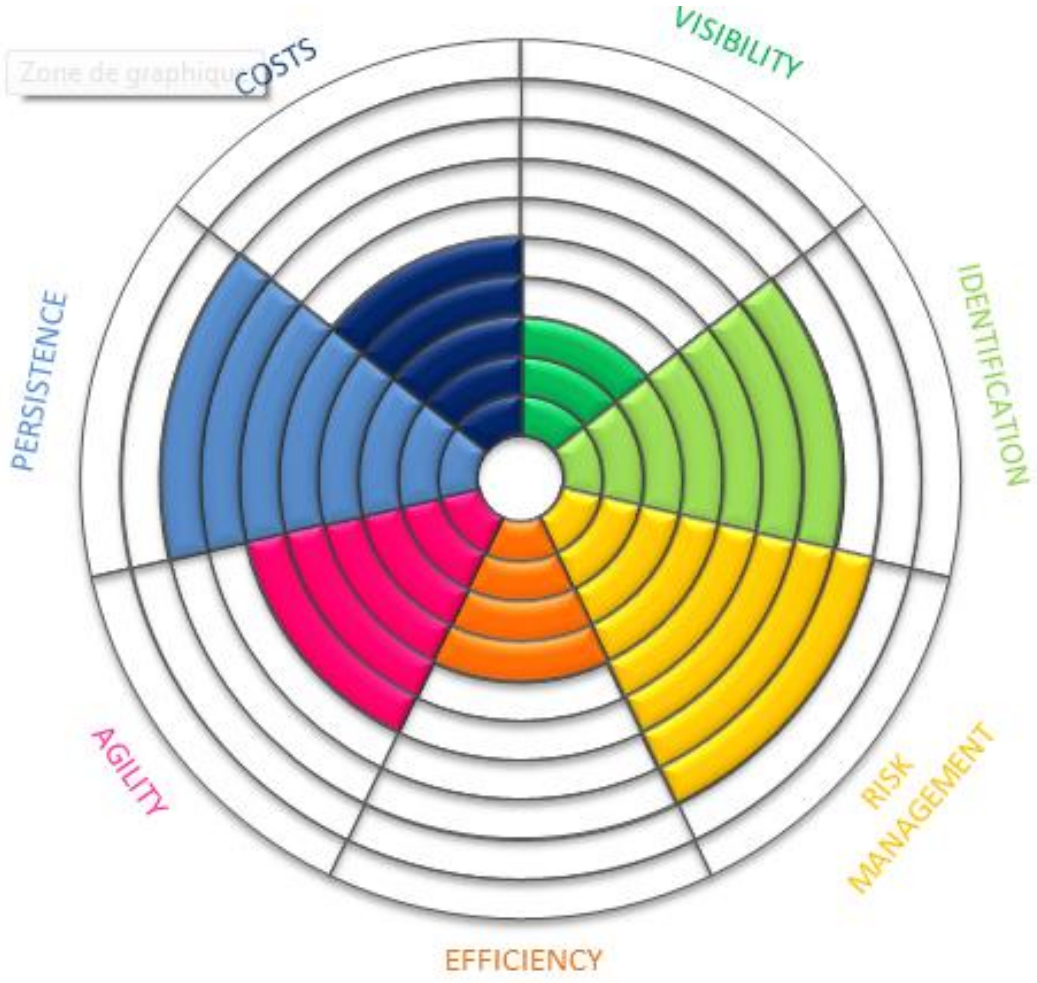

Figure 12 - Eracent Evaluation Summary

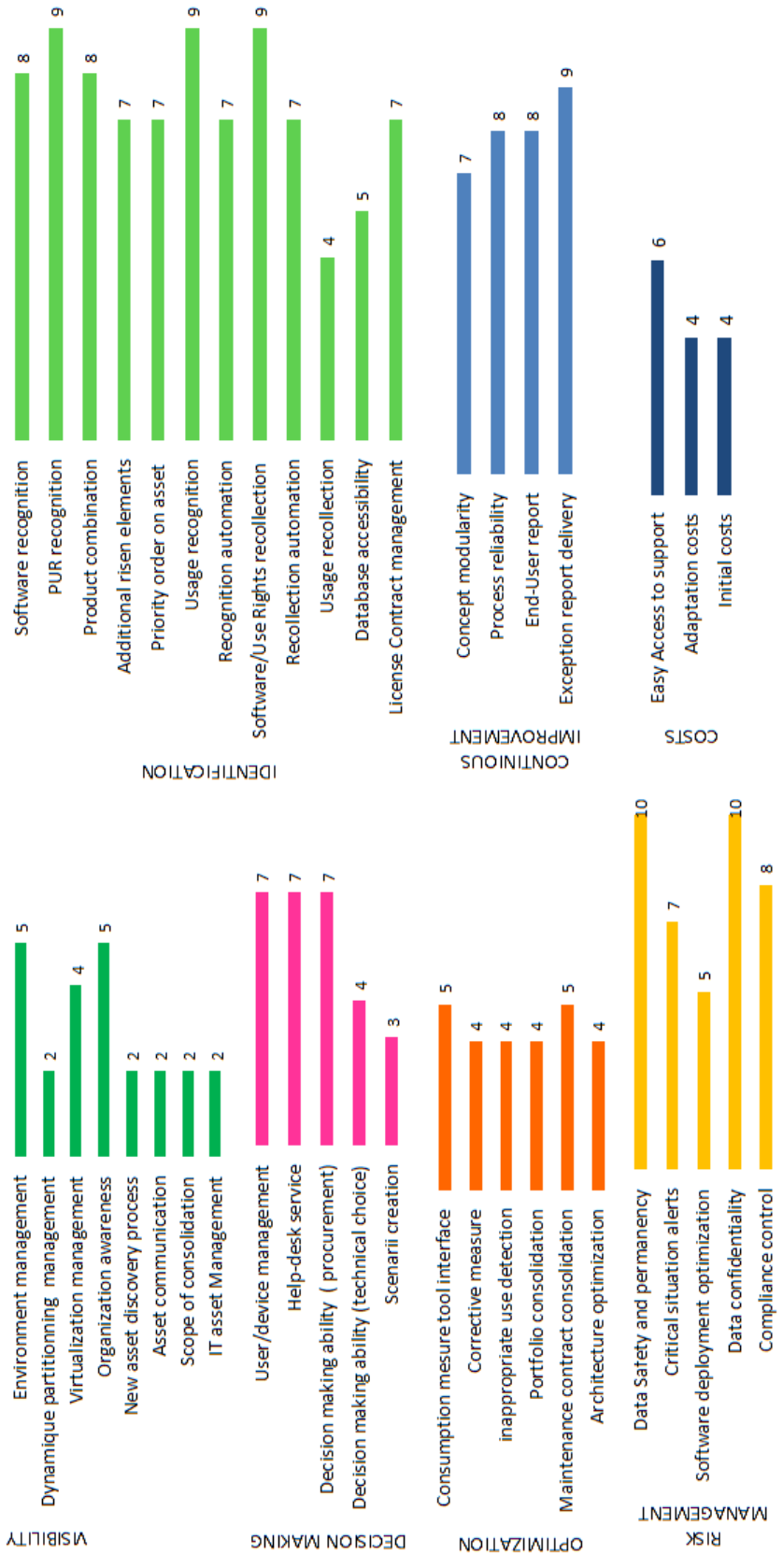

Figure 13 - Eracent Evaluation detailed view

f. HP Asset Manager

HP Asset Manager offers a quite strong asset management discovery tool with loads of possibilities if you want to have a picture of each asset attached. The functionality where the scan agents are pushed out to the clients works well.

Basic recognition is quite poor, and the process of adding/learning new software is complex and time consuming. Focusing on Software and Compliance HP Asset Manager is really weak. There is no report builder, so you either stick to the basic reports, or need to invest in more developments. A lot aof home-made development are required, and the design and usability looks like something from the 90's.In general this tool is too complex for the non-advanced user and SAM module is difficult if used with external discovery sources.

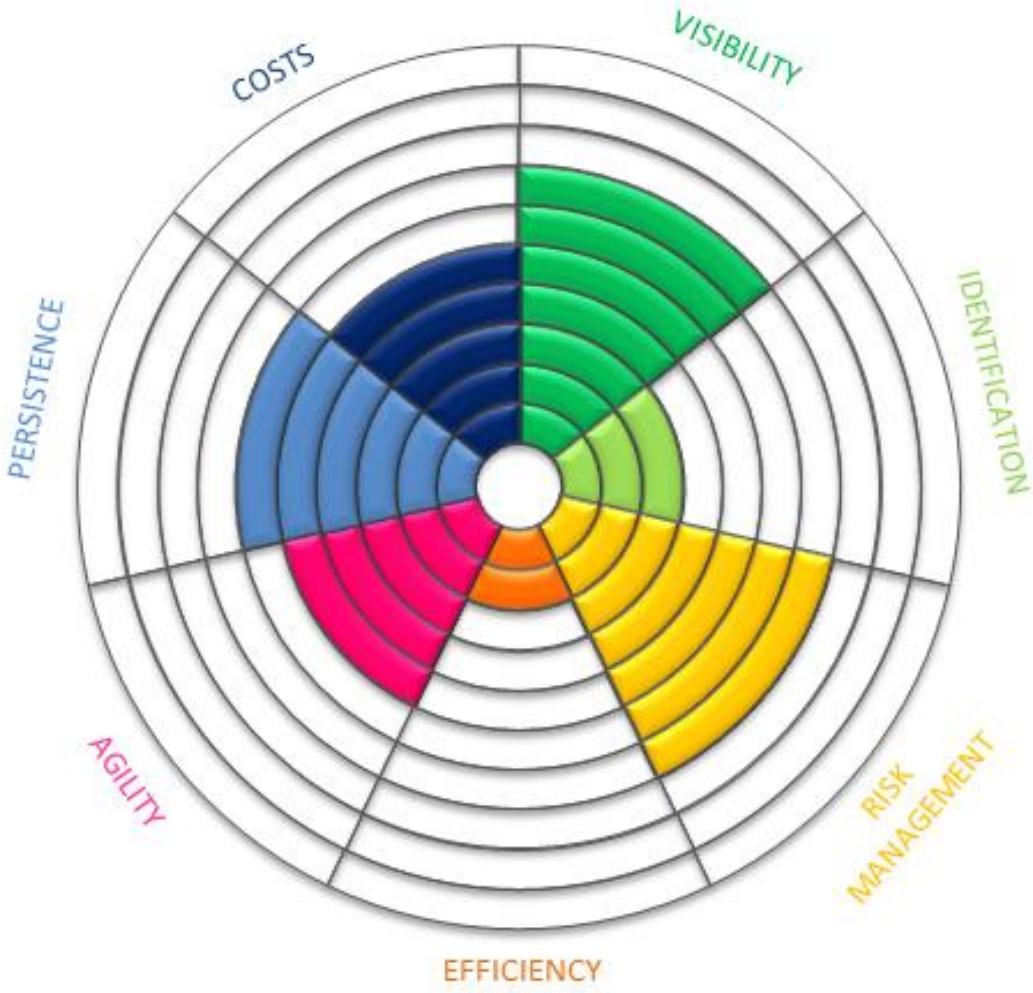

Figure 14 - HP Asset Manager Evaluation Summary

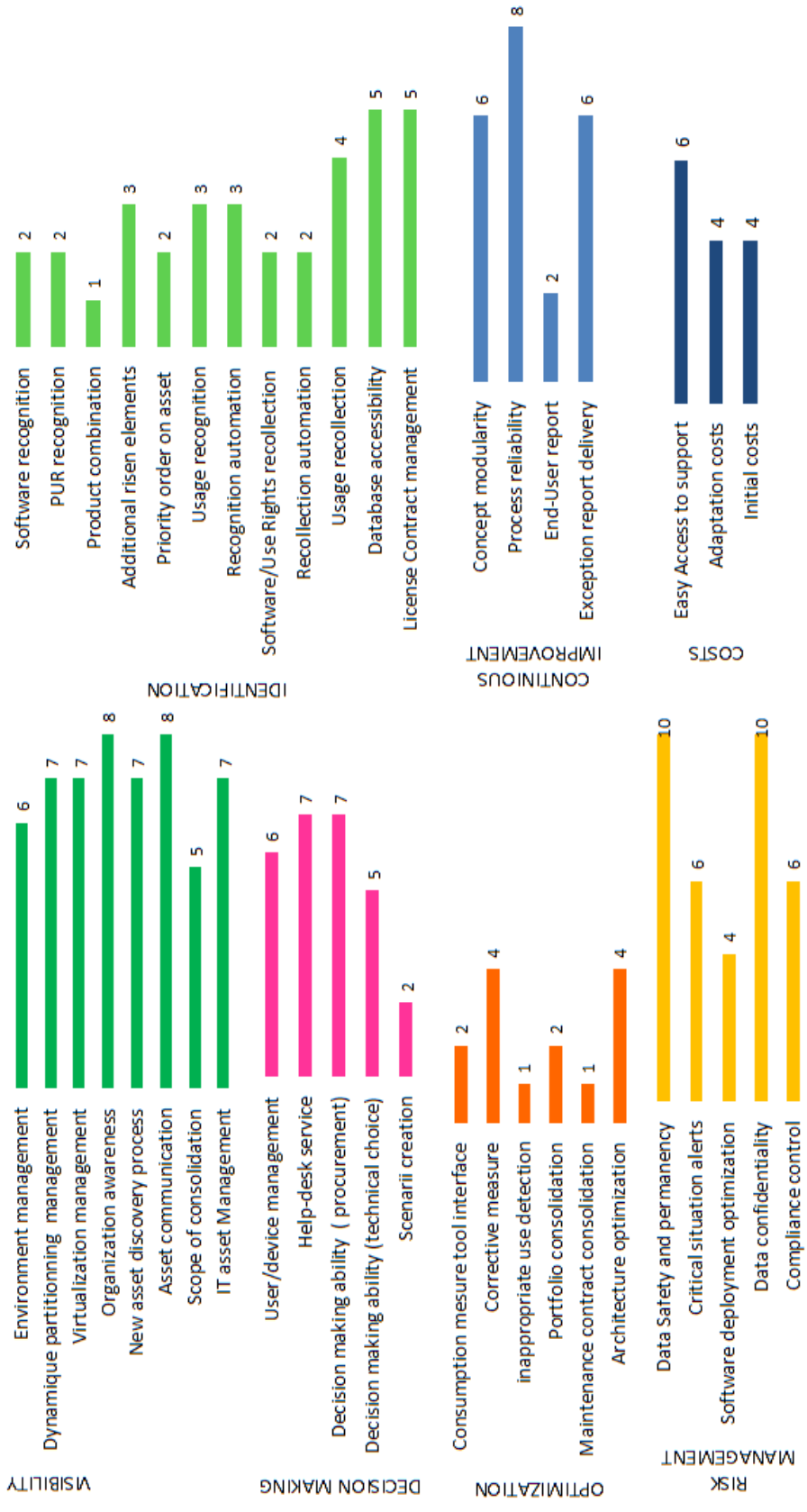

Figure 15 - HP Asset Manager Evaluation detailed view

g. BMC Remedy

BMC offers solid asset management principles properly applied on a good workflow and coverage across all elements of asset management (mainly geared for hardware and basic software compliance though). It also integrates to the wider Remedy CMDB, so will tie an organization incident and problem management system and configuration management system with the asset management system which is really the key strength of using this as the Remedy suite.

Remedy is not focused on the deeper software analysis that is now available in competitive products leaving organisations to fill the gap themselves, or through 3rd party services. The product also requires a large amount of 3rd party services to keep it running. Remedy is not intuitive to administer (either back end or front end) and splits the deeper information to Atrium Discovery module which means you end up using two products to get good reports. It seems that it has been left behind by the other SAM competitors.

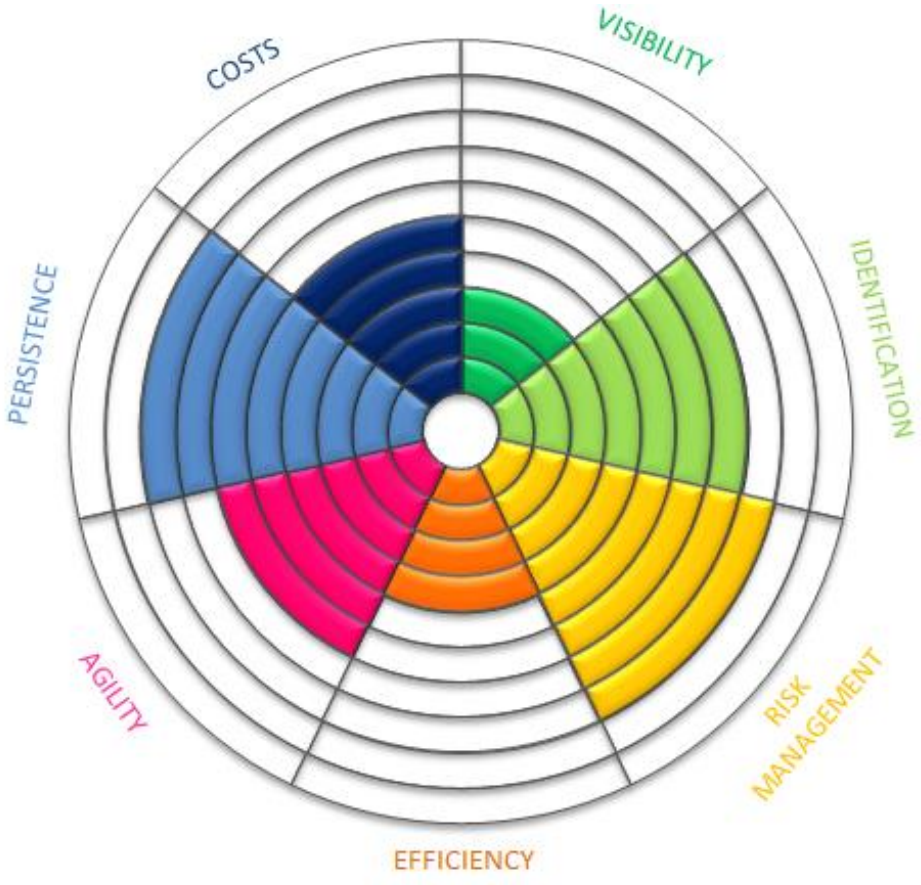

Figure 16 - BMC Remedy Evaluation Summary

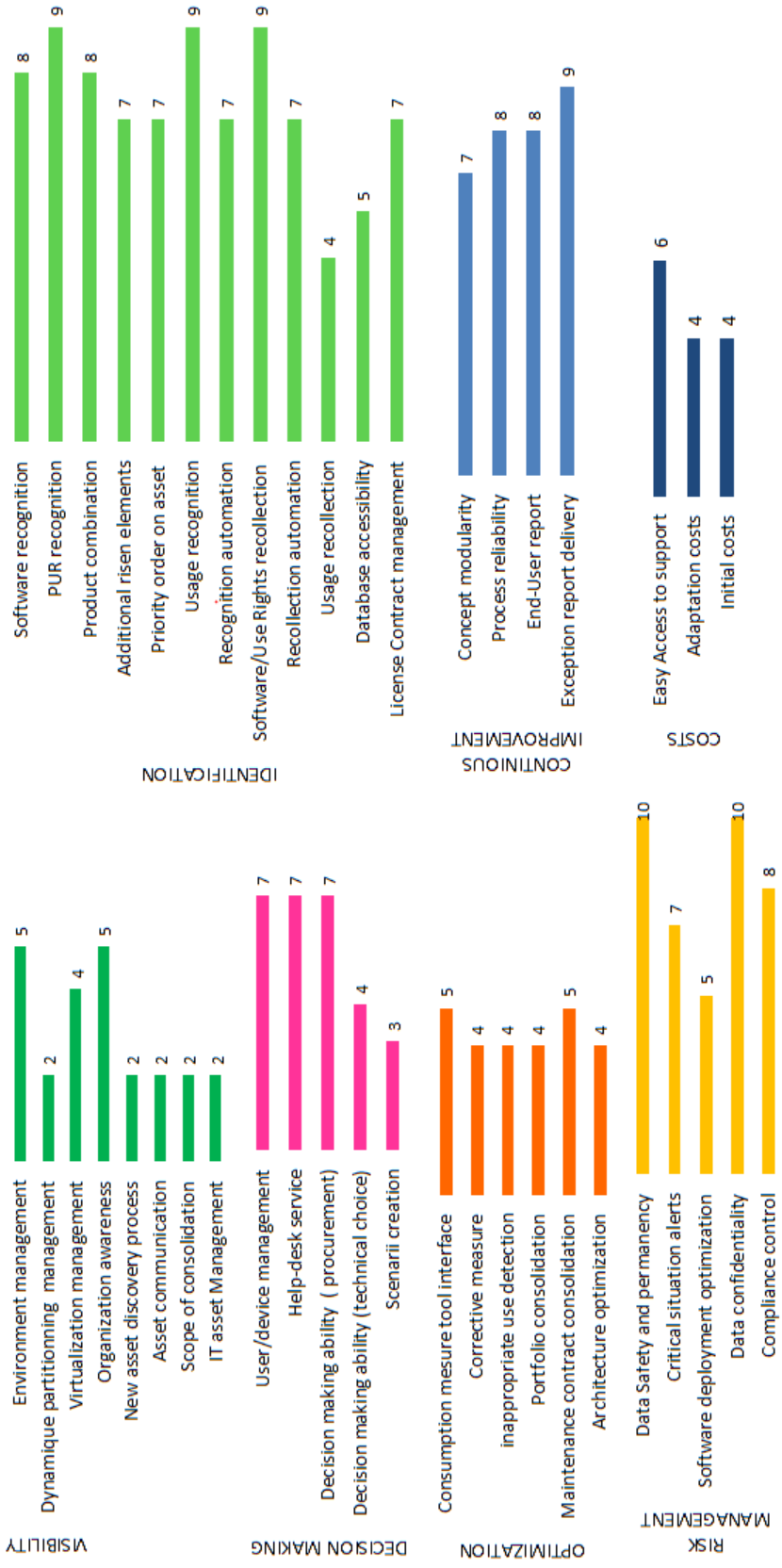

Figure 17 - BMC Remedy Evaluation detailed view

h. GLPI – OCSng

The couple GLPI – OCSng offers a quite strong asset management discovery tool with loads of possibilities if you want to have a picture of each asset attached. The functionality where the scan agents are pushed out to the clients works well. The injection in GLPI for inventory overview is interesting. Yet, virtualization recognition is quite basic and the process of adding/learning new software is complex and time consuming. Focusing on Software and Compliance GLPI is really weak. There is no report builder, so you either stick to the basic reports, or need to invest in more developments. Alike, there is no dedicated SAM module and no automatic license stock review. A lot of home-made developments are required (we underline the open-source license of GLPI) to enrich licensing modules, and the design and usability are not easy to handle. In general data injection is difficult if used with external discovery sources.

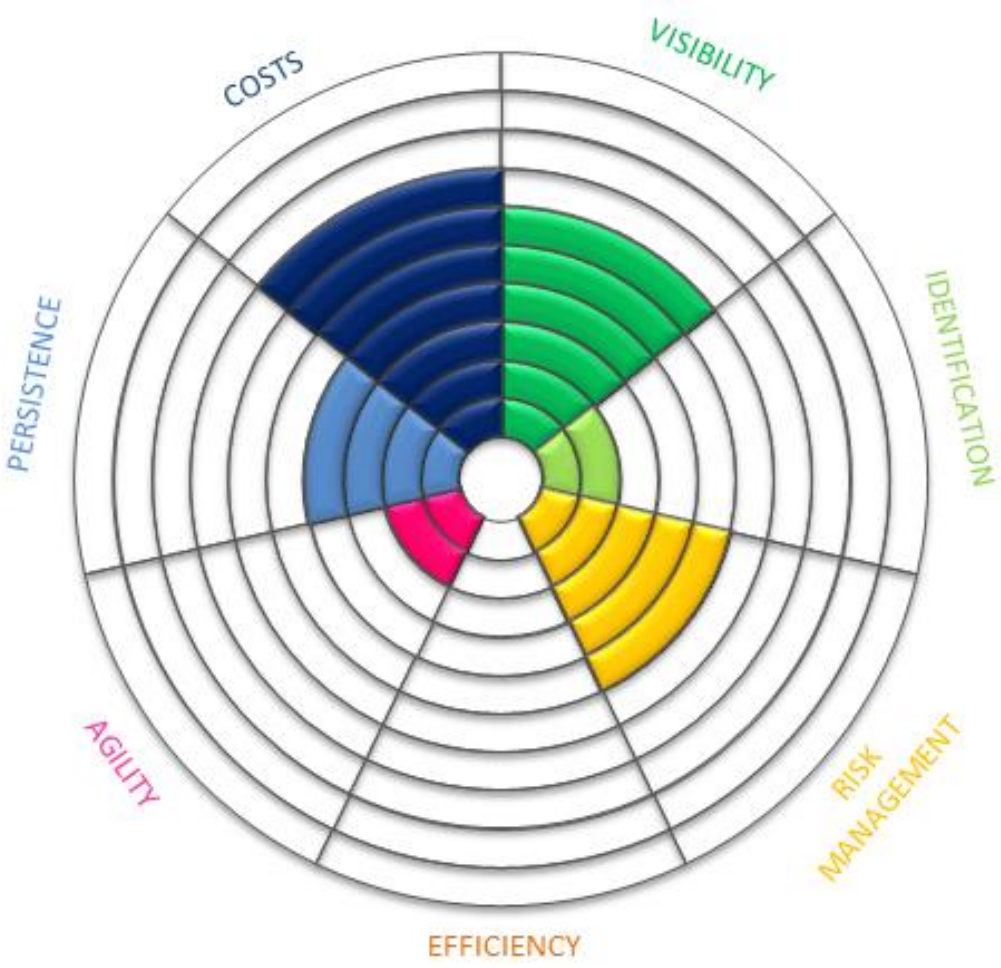

Figure 18 - GLPI/OCSng Evaluation Summary

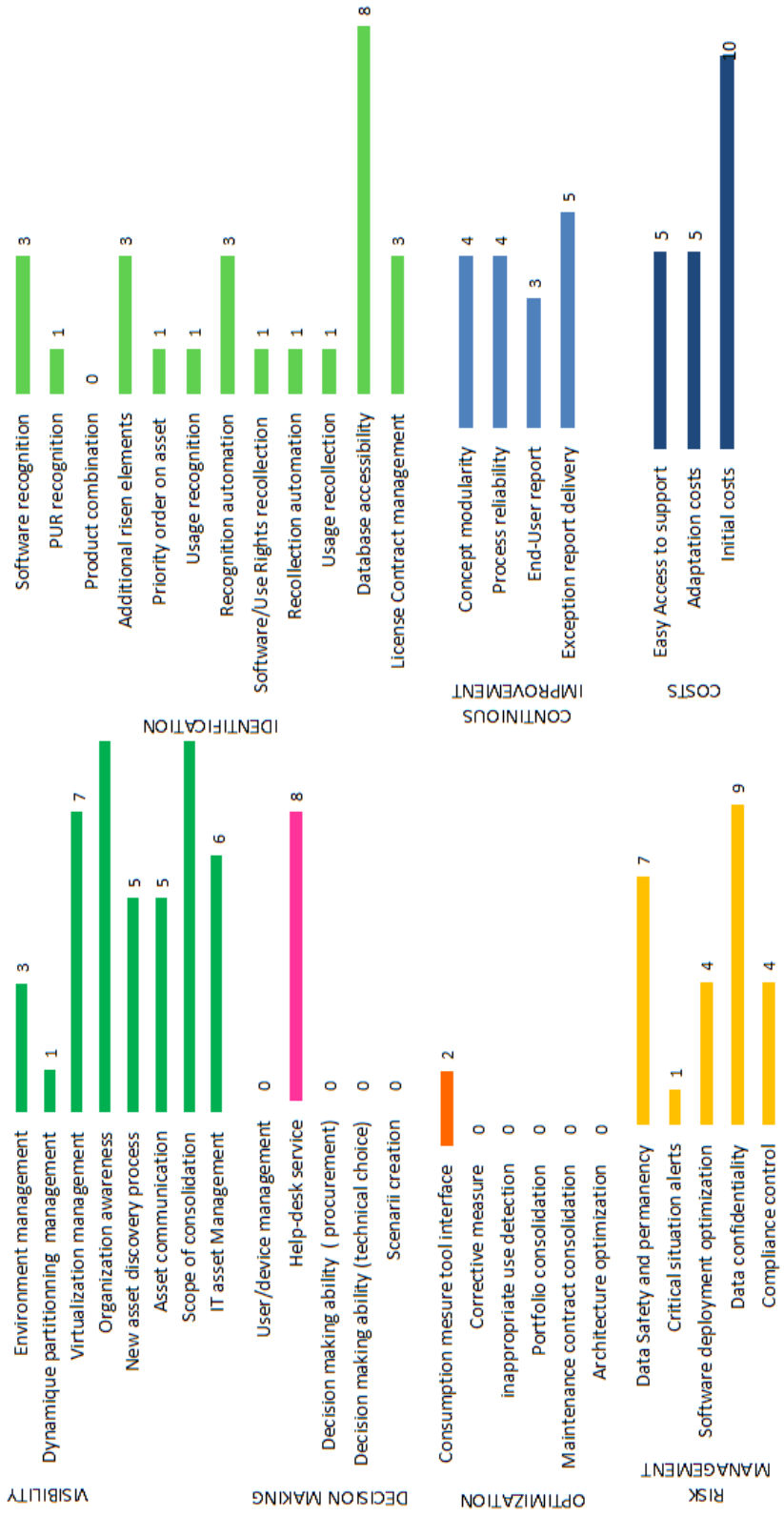

Figure 19 - GLPI/OCSng Evaluation detailed view

2.5. SYNTHESIS ON INDUSTRIAL STATE OF THE ART

After identifying on horizontal axis, the prospect for improvement of SAM processes and on horizontal axis, the complexity factors brought by the cloud, the focus should be done on weak points of SAM processes. (Fig.20) summarizes the evaluation of major market tools. We propose to use the SAM maturity scale to read it. Visibility is first step and mainly we will find discovery tools (BladeLogic, OCSInventory NG, and SCCM[12]). As transition to the second step: Identification, we will find tools like GLPI, to manage assets discovered in first step but without being able to truly identified software like tools proposed by Aspera, snow or editors' own solutions able to manage PUR and for some able to identify risks of over/under deployments (Snow, Spider Brainware group, Aspera).

It summarizes that real sticking points for the expansion of SAM cloud management are mainly based on level two of the SAM maturity scale presented above: identification of software and modeling of automatized policy and controls to sustain dynamic and real-time cloud provisioning. We propose to address these two points in section 3 and 4.

Figure 20 - Features and limitations of most popular SAM tools

[12] www.microsoft.com/fr-fr/server-cloud/products/system-center-configuration-manager/, September, 2016

3. SYNTHESIS

Going further, in cloud environments SAM is not only assets management, but also service management which must be done in real time taking into account the fast rhythm of changes: services are provisioned, configured, reconfigured and decommissioned in a matter of minutes (summarized in Fig.21). Compliance risks are increased by the ease and speed of provisioning which can bypass traditional centralized processes. In such conditions, SAM controls are difficult to implement.

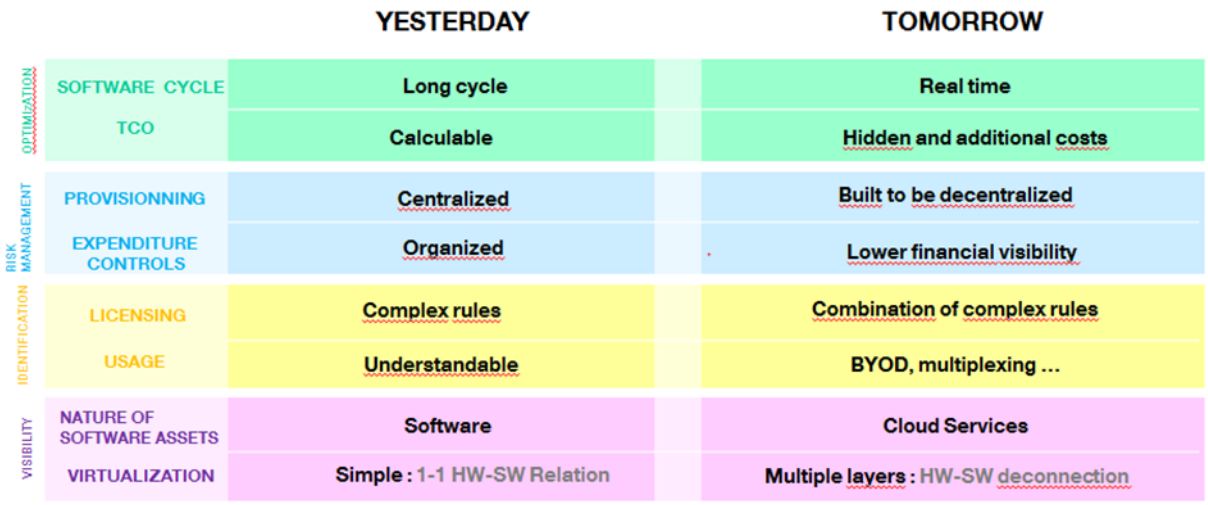

Figure 21 - Complexity factors brought about cloud architecture

One of the business benefits of cloud computing is its agility and speed-to-market. Services are provisioned, configured, release in a matter of minutes. Thus, while traditional SAM processes assume long lifecycles (usually, we can consider 5 – 7 years for a software, which leads to long cycles of contracting, discovery, inventory and reconciliation), cloud is accelerating these processes up to real-time requirements.

A second issue to consider with cloud environments is that different levels of services and multiplication of hidden costs have to be taken into account. These hidden costs may include cost of migration, integration with IT systems, premium support services, new storage requirements, cost of extraction of data, renewal costs of the service, oversubscription costs.

We can also underline that if SaaS seems to reduce or even delete infringement risks because it is supposed to be indexed on real usage, this use is in fact restricted in many cases and is not often negotiable. In such cases, SAM should have proper controls in place to ensure compliance with all requirements and

limitations (geographical scope, restriction on shared accounts, on non-employees/providers, partners … time of day, volume of transactions …). It leads to multiplications of complex rules, not only based on hardware metrics, but directly on usages, sometimes more difficult to identify.

As said in (BSA, 2014) [26] cloud services are often considered as operational expenses and not as capital expenditures, which can lead to several problems: (1) less involvement in contracting phase, (2) loss on control of operational dependencies, (3) loss of know limits to final costs, (4) lack of financial visibility, (5) increased license compliance risks.

Chapter 3

III. Proposition of a Software Identification Model for Cloud Environments

Proposition of a Software Identification Model for Cloud Environments 73

1. Software licensing issues, challenges and opportunities 77

A. Licensing complexities moving to the cloud 78

B. Cloud-oriented software licensing models 81

2. Requirements for effective Software and entitlements identification 81

A. Requirements for Software identification 82

B. Requirements for Usage Rights (PUR) or entitlements identification 82

01. Deployment and migration conditions 82

02. Access & usages conditions 83

03. Geographical and location requirements 83

04. Elasticity scope 83

C. Requirements for Instances and deployment environment identification 84

D. Requirements for identification inherent processes 84

01. Impact on procurement 84

02. Impact in measurement and tracking usages 84

3. Software PUR management process flow 85

4. Implementation of software identification patterns 88

A. Software identification hindrances 88

B. Software identification models proposed by ISO 19770-x 90

01. ISO 19770-2: about Software Identification Tags 90

02. ISO 19770-3: about Software Entitlements Tags (Ents) 96
5. Workaround propositions and use-cases 98
A. About PUR 98
B. About SKU 99
C. Proposed identification Lifecycle 101

In this chapter, we will (1) detail the software licensing issues, challenges and opportunities brought by the cloud; (2) expose requirements for the effective software identification, including entitlement's identification; (3) propose a software entitlement management process flow and (4) discuss about implementation of software identification patterns.

Software business is often complicated by use of unprecise jargon and acronyms. To help classify matters we propose here three definitions to make a distinction between Software license, Software key and Software entitlements which are commonly misunderstood and will be developed all among this chapter.

- A software *key* is a special piece of software that unlocks the product and allows it to run. Many vendors incorrectly refer to keys as "licenses". Moreover, contrary to the name, "license servers" do not actually manage licenses but keys and do not show users how many licenses a server manages. To re-use a well-known real-world analogy: if you owe a house and it key but lose your key, you are still owner of your house. Alike, if you found your neighbor's key, it does not mean that you owe his house. License grants a user the right to use the software. Holding a key is not equivalent to owning a license, just as having a door key does not make you the homeowner.

- A software *license* is what grants a customer the right to use a specific product. It contains a set of terms and conditions (in other words called Product Usage Rights (PUR)) that define to what extent you may legally use that software. When taken on its own, however, a license only provides enforcement via legal recourse.

- Software *Entitlements* and *PUR* represent software use rights granted by a license as defined through agreements between a software licensor and a software consumer. Entitlement management is a system by which rights are assigned to their intended recipients and then managed. It provides fine-grained management over the rights to use the license and, as a consequence, the software. It enables you to grant, resolve, enforce, and revoke access entitlements, as well as enforce access policies for data, devices, and services.

Moving to the cloud is not going to simplify license lifecycle's management especially because of complexities on software entitlement management. These hindrances affect among others, cloud providers, cloud subscribers and software

vendors and require cloud deployment dedicated solutions. To operate in this environment, software users from cloud providers to cloud subscribers must manage their PUR while balancing the usage, price, and performance features of software entitlements with the software licensors. We can consider the two following hindrances:

- There is multitude of software vendors proposing multiple of different licensing schemes increasing the complexity of managing software entitlements. One product might be distributed under 1 to n* different metric(s). One metric might have different meaning depending software vendors. Entitlements encompass large variety of limitations and effective use rights take into account any contracts and all applicable licenses, including full licenses, upgrade licenses and maintenance agreements. Two relevant examples of this multitude of licensing models: in 2015, IBM was proposing 143 different active IBM license metrics[13], only one could be tracked via SAM tools (Processor Value Unit (PVU)). The German software editor SAP had 70 different active license metrics.

- From traditional to hybrid and complex software entitlement management structures, new specific mechanisms must be implemented to overcome cloud deployment's complexities.

In such context, predicting the total cost of software - including licensing and managing compliance - present growing difficulties. Within a single PaaS or software-as-a-service (SaaS) environment, multitude of entitlement models and metrics exist for the different components and have to be synchronized and reported with real-time level of requirement. But first of all, entitlements have to be properly and quickly identified.

The objectives of this chapter are:

- To analyze Software licensing issues and challenges in Clouds
- To discuss about minimizing risks of software non-compliancy through a detailed process of PUR identification, including requirements and specifications.

[13] D.Foxen, May 5th 2015, report on IBM & SAP Seminar. (online) https://www.itassetmanagement.net/2015/05/05/ibm-sap-seminar-london-april-2015-report. October 2017

- To discuss about relevancy of software identification pattern and existing commercial initiatives and hindrances to standardize a software identification model.
- To propose a cohesive identification model to accommodate scalable and dynamic cloud deployments

It will be organized as follow: (1) Software licensing issues, challenges and opportunities in cloud environments; (2) Requirements for effective identification of Software entitlements; (3) Software entitlement identification process flow, (4) operational software identification regarding current norms and practices.

1. SOFTWARE LICENSING ISSUES, CHALLENGES AND OPPORTUNITIES

In traditional architectures, we were used to see PUR tied to specific computers, servers, resources (CPU, disk…) or users. This specificity does not fit to cloud deployment models where the cloud subscriber looks for capability to dynamically dimension software, as needed, without real-time compliance concerns. In other words, the cloud significantly complicates the effective management and optimization of software entitlements for cloud subscribers, vendors and providers.

As an intangible asset, it is difficult to evaluate fair price of software and what can be fair licensing costs. It can explain the gap between software consumers and software vendors approaches to fair licensing and it requirements. It explains partially the diversity of software licensing and pricing models being demanded by carriers and offered by software vendors.

Software vendors propose multiple licensing models driven by:

- Increasing the predictability of their revenue. Software vendors are expecting software licensing and maintenance revenue predictability when software consumers are expecting predictability of the licensing costs
- Better understanding of their customer software uses. To increase the value of product and maintenance services directly bound to customer needs
- Indexing software value on participation in wider solutions. Improved alignment with value. Demonstrating the tangible value of intangible software through relevant proof and metrics can improve alignment

between licensing costs and both editor and customer's perception of software value.

Software users expect software vendors to:

- Improve the effectiveness of licensing practices
- Allow flexibility and simplicity when proposing software licensing contracts. License term's complexity is directly linked with non-compliance or accidental piracy.

1.1. LICENSING COMPLEXITIES MOVING TO THE CLOUD

Most often-used 'traditional' licensing models (such as number of cores, CPUs, allocated physical resources, etc.,) bind software deployments to physical infrastructures or hardware features (ownership, geographical restrictions, installations, etc.,). This binds between IT environment and software licenses are limiting usage and capacity especially when migrating from traditional IT models to flexible cloud infrastructure. Actually, traditional granted usage-rights do not match with cloud requirements such as virtualization, elasticity and on-demand. In virtualized environment, an issue consists in mapping physical licensing to virtual resources. It might be difficult to have the same use of software for equal costs. The issue is nearly the same considering mobility between private, public, hybrid, multi-tenant clouds which also implies software entitlements changes and compliance failure.

Supervision of licensed software consumption is more difficult given the increased complexity of identifying and tracking compliancy issues. Dynamic provisioning of instances might lead to compliancy issue like underutilization or overuse of assets without possibility to counterbalance it. Actually the ease of migration and instance cloning force to multiply tracking and matching on multiple platforms, data-centers, private/public/hybrid clouds across more complex software lifecycle, to faster time-scale. Migrations and resource allocation changes across datacenters and deployments weaken compliancy and accuracy of entitlement inventories. The ease with which resources can be dynamically allocated and used (scale up or scale down) in virtualized environment causes issues to predict the initial and ongoing cost of software licensing. Hybrid license models that encompass usage and device-based licensing models increase the risk to burst limits and breach PUR agreements.

Considering the most commonly used metrics (processor, devices, user, access), we can list (and summarize in Tab. 3) some major risks moving these licenses schemes to cloud infrastructures.

- Bound to the **processor capacity**, like CPU (Core processor Unit, from Oracle), PVU (Processor Value Unit from IBM), Core, processor, etc.
    - In traditional architectures, these metric, especially in virtualized environments are often complex, slightly different from one to another depending the editor. Keeping track of the proper amount of processor license counts and capacity levels typically requires deployment of advanced monitoring systems. Moving to the cloud, monitoring systems to track processors counts and capacity levels in IaaS can be more challenging due to compatibility, security and network issues.

- Bound to **devices** proposed by most publishers, like Instance, Device, Computer, Installation etc.
    - Often in traditional architectures, discovery tools and delivery processes reduce risks of non-compliance for device licensed products. As good is your coverage, as lower is the counterfeiting risk. In cloud infrastructures, software discovery is more challenging, due to diversity of technologies and cloud (non)-interoperability, levels of security and monitoring. Additionally, considering SaaS, many products can be accessed and used via multiple devices; thus, keeping track of licensable devices can be challenging

- Bound to **User** proposed by most publishers, like Standard/Professional User, Limited User, Administrator/reader etc.
    - In traditional architectures like in Clouds, usage rights for each user role are tailored in software license agreements. Access to usage rights can hardly be technically restricted, and are difficult to report and translate into licensed roles when Cloud demand real-time visibility on user's usage right assignments.

- Bound to **In-direct Access**, proposed by many publishers, especially by SAP, IBM and Oracle, like Named User, Authorized User, Employee, etc.
    - These licensing rules often call for all interactions between software and human users either directly, through a named account, or indirectly through a shared account or third-party

application account, to be fully licensed. For SAM purposes, it is hard to obtain more than the visibility of the number of accounts (and not the true number of user behind each account) within an application which is not showing the true amount of access. Obtaining such visibility is again more challenging in cloud environments due to the difficulty to fully observe the system architecture and the multitude of user access mechanisms.

Metric	From traditional architectures	To cloud architectures
Processor	Different licensing terms between vendors and difficult to understand. Requires advanced monitoring systems to track resources	Temptation to paid for virtual capacity while physical capacity needs licensing Track and monitoring even more challenging.
Devices	Risk is limited because software discovery has quite good coverage	Lower discovery performance. Multidevice access
User	Bespoke User's rights No technically restrictions due to difficult translation	Usage real time visibility requirements
Access	Require difficult full visibility on all access (direct and in-direct) and accounts	Less visibility on system architectures and access.

Table 3 - Most used metrics and identified risks in cloud environments

This being said, we stress the necessity to create dedicated licensing models and specific contractual terms for cloud environments; to simplify entitlement identification in order to ensure compliance management, support cloud deployment flexibility and dynamicity and to gain a better understanding of contractual terms used within the license scope (i.e., does processor mean CPU or core? "Named user" include or exclude batch processing?).

Nevertheless, moving to the cloud should not mean for cloud-service providers, that their license legacy becomes obsolete. Perpetual licenses bought for traditional architecture should not have a practical limited lifecycle due to technical obsolescence of the IT environment. A perpetual license is an entitlement for an unlimited period of time which cannot be bound to the current technology but be adapted to fit it.

1.2. CLOUD-ORIENTED SOFTWARE LICENSING MODELS

To better suit cloud deployments and their flexibity requirement, software vendors started to propose new licensing models or to define adjustments to existing ones.

Thus appears SaaS subscription-based licensing models (or 'pay-as-you-go') where the license consists in a subscription basis depending on the number of users. The SaaS provider manages accountancy, underlying software components (Operating system, middleware, etc…) and the consumer is responsible for auditing and monitoring compliance. However, in IaaS or PaaS environments, issues described above remains unsolved is case of legacy license asset migration from traditional to cloud infrastructures. New pay-as-you-go models might be interesting for some new services but do not allow to use with flexibility, already-owned licenses. Some vendors propose new entitlements bound with virtual allocated resources (vCPUs, virtual appliance, etc.,) or virtualized environment sizing (i.e based on a virtual network element number) which better fit to cloud infrastructure but force to adapt tracking on multiple platforms and clouds across more complex software lifecycle, on real time identification requirement.

2. REQUIREMENTS FOR EFFECTIVE IDENTIFICATION OF SOFTWARE AND ITS ENTITLEMENTS

The characteristics of software covered by a license, its instances and their consumption have to be traceable regardless of the deployment (physical or virtual computing through a virtualized environment, from the cloud and through a data center). This requirement of traceability encompasses precise identification of (a) software, (b) its PUR or entitlements, (c) its instances. The software vendor entitlement requirements will impact the software user entitlement management processes, in particular:

- The features implemented to manage and monitor software PUR at a consumer level using SAM approaches
- Maintenance of deployment inventories
- Usage-based reporting of software deployments
- Internal elasticity management policies to fulfill entitlements requirements

2.1. Requirements for Software identification

Key attributes of software include its commercial name, version, editor, third-party vendor, eventual patch and release and their version; It encompasses the identification of packaged software/application: licensed software products can be packaged to form solutions, suites, bundles, and virtual appliances.

2.2. Requirements for Usage Rights (PUR) or entitlements identification

A catalogue of PUR should be implemented to store entitlements grouped by software. It should encompass the term of the license and termination provisions, including post-termination transition rights; Licensing metrics and model used, such as named user, concurrent license, volume license, enterprise/personal license, evaluation/trial license, original equipment manufacturers, hardware platform- or device-based, role-based, employee-based, financial-based, or transaction-based; and usage rights and restrictions like deployment restrictions, including geographical restrictions.

Such catalogue of PUR should be integrated in an audit process of license limitations especially for migration purpose, but not only. It should assess among others (1) deployment and migration conditions, (2) access & usages conditions, (3) geographical and location requirements, (4) Elasticity scope.

a. Deployment and migration conditions

We should identify if software can be deployed across the physical and/or virtual infrastructure if migrations to the cloud are permitted and how?

- From physical data centers to virtual data centers
- From a virtual host to another virtual host within a virtual data center
- From one host to another host within a public cloud
- From one host to another host within a private cloud
- From a virtual data center to a public cloud, and back
- From a private cloud to a public cloud, and back
- From a public cloud to another public cloud, and back)

The natures of deployments, migrations, cloning varies depending the goal and has to be easily identifiable. Adequate identification should be in place to identify software vendors policies in case of back-up, standby equipment to contend with hardware failure, parallel maintenance tasks to facilitate workload shifting, load balancing in order to maintain service quality. As well, software vendor position toward cloning of virtual machines has to be easily known.

b. Access & usages conditions

It encompasses limitation of the number of cloud users (people, human operated device, non-human operated device, application etc..) who will be allowed to access software, and clarification on how are considered the different type of access (i.e. does "access" include or exclude batch processing?). Access rights should be clearly defined in the context of user types and business need.

We should be able to identify some restrictions which can be based on the type of environment, depending if the software instance will be used in development, test or production or in combination of independently developed and supported products. The impact of routine maintenance performance should be assessed. As well, we should identify if any part of the cloud application delivered by the vendor is outsourced or subcontracted to some other third parties.

c. Geographical and location requirements

Some software vendors limit where the software can be deployed or used through software entitlements. We should be able to identify the restrictions on geographical locations where a license can be used and the instance can be provisioned and offered. Access rights should be clearly defined and managed in the context location of access.

d. Elasticity scope

We should be able to identify:

- If the license can support cloud bursting or migrating from one cloud to another and how to consider movement of cloud services and VM in order to balance the data center load, to support disaster recovery, to handle data center migrations, to handle capacity burst requirements?
- If the license allow transitory use of specific software
- The level of elasticity granted by licensing quotas and burst limitations should be identified and assessed.
- If the vendor propose an elastic infrastructure with defined limits that will ensure software entitlement obligations

2.3. REQUIREMENTS FOR INSTANCES AND BOUNDED DEPLOYMENT ENVIRONMENT IDENTIFICATION

Basically, it includes whether software is currently used and the ability to map software deployment back to its corresponding PUR and provisions for accommodating legacy software PUR, such as those that correspond to physical-hardware deployment.

Regarding packaged software, we should have interoperable capabilities to discover bundled software within the deployment package in order to automate and manage software installation, audit and migration.

2.4. REQUIREMENTS FOR IDENTIFICATION INHERENT PROCESSES

a. Impact on procurement

The level of identification requirements should impact the role capacity from companies (software user) to procure software. In other words, it defines if employees can procure software directly through Internet download or the level of procurement department centralization. Likewise, it impacts the process of software deployment and migration after within the company (who allows it and where?) and how software license fees will be paid and through which channels? A centralized procurement reduces counterfeiting risk exposition by eliminating the acceptance of hazardous contractual terms, impracticable license usage rights and restrictions, and not suitable financial costs.

Identification of entitlements should also impact the relation between procurement and IT department for such questions like approved commercial consideration regarding IT needs. For example, to balance more cost effective metric with relevant IT deployment requirements. Companies should assess the commercial goals of the agreement, evaluating the anticipated workloads for normal and extraordinary short-term and long-term business use-cases. Then, they should review which type of licensing patterns more are acting to meet their needs, including interoperability and commitment requirements.

b. Impact in measurement and tracking usages

These requirements are directly bound to those expressed in previous chapter and constitute prerequisites to implement processes and tools to precisely identify and monitor usage for any software instance. Identification of entitlements should allow a cloud subscriber to set up relevant SAM processes like:

- Build and update software entitlement ad usage library taking into account interdependent software delivery processes

- Monitoring license compliance and implementing controls to manage licensing compliance violation.
- Set up internal and external optimization of software licensing investments
- Evaluate accuracy of provider reporting capability and reciprocally
- Propose relevant metrics to measure as precisely as possible real consumption of software, based on metric's costs and benefit comparison
- Point suitability of software entitlements regarding their ability to be measured or their cost-effectiveness

3. SOFTWARE PUR MANAGEMENT PROCESS FLOW

The integration of software PUR management and its consequences across the software lifecycle are described in Fig.22. Once a contract (commercial agreement) has been set up with a software vendor, entitlements are controlled through a set of key operational processes within the whole lifecycle described in previous chapter. Briefly, we can count:

- *Service Catalog* creation and update which consist in maintaining a set of available services and levels and provide real-time information about available stocks and applicable prices and conditions. The catalog lists all proprietary software that requires licenses at company level and special contractual agreements and arrangements.
- *Image Catalog* which consist in maintaining a library of VM and software image enriched by entitlement metadata in keeping with service catalog.
- *Provisioning* consist in charge required workload to available environment respecting given limitations and permissions
- *Identity management* consists in federating identity sources, prerequisite for user-based software licensing
- *Delivery and operations* consists in installing and accessing software according to given entitlement limitations and necessitate to update software entitlements database. Other activities can be provoked by internal or external events like bursting
- *Monitoring* consists in detecting events in installation or usage of software regarding bounded entitlements

- *Metering* consist in registering actual usage of software components according to metrics which are bounded to software
- *Billing* consists in arranging payment for used software resources directly to software vendors or third-party
- *Termination* consists in terminating services when no longer needed and analyze their consequences in terms of entitlements in case of usage-based licensing mainly

Each stakeholder has its own set of interactions. The processes follow a logical order, although some of them are called iteratively, and in some cases there are more complex patterns involved between steps. Software supplier cans SaaS, PaaS or IaaS suppliers.

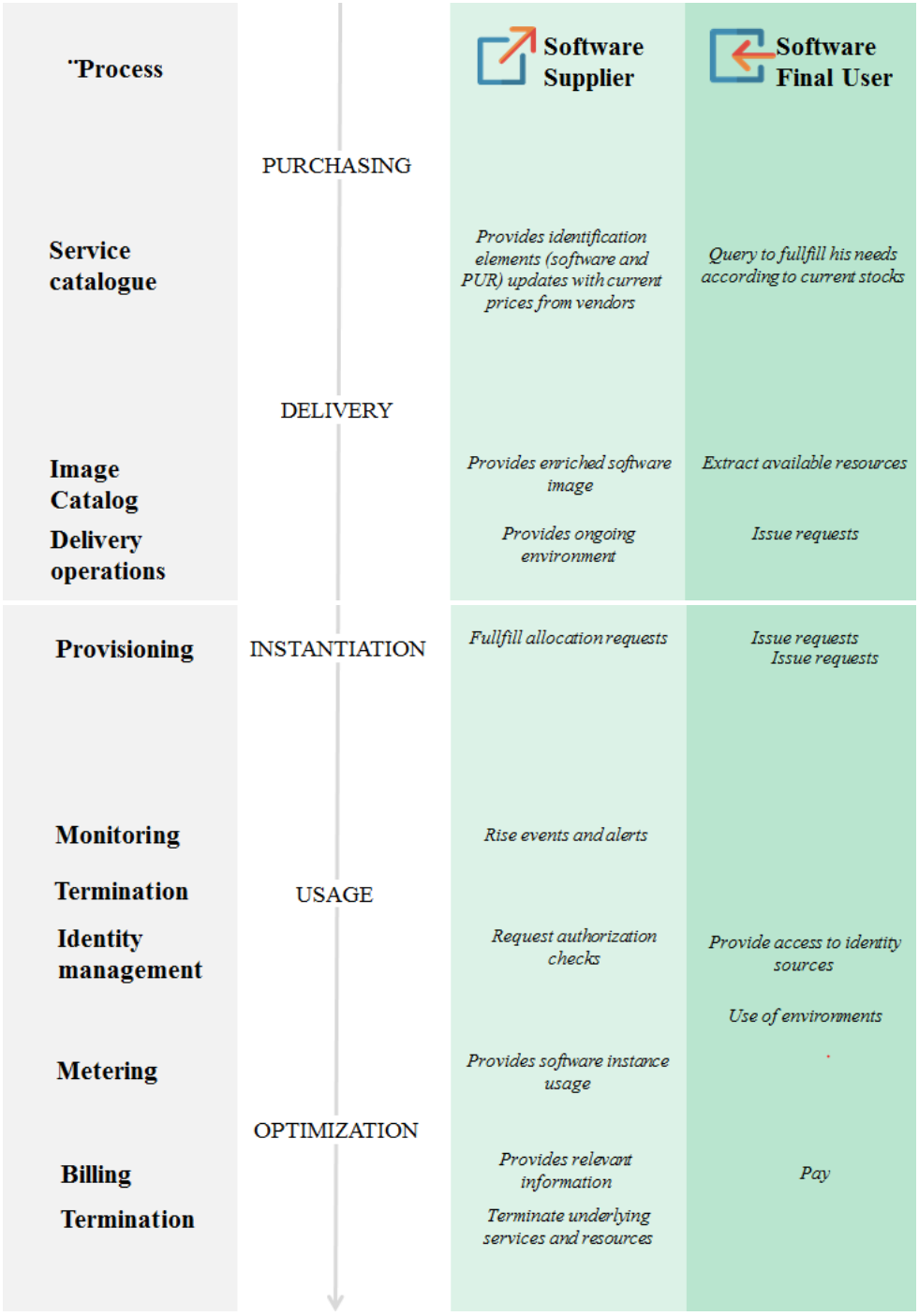

Figure 22 - Key operational processes in software PUR identification

Some interactions are possible between the processes which are not only following vertical axis. For example, provisioning might be provoked by some terminations and billing is triggered by metering and usage.

4. IMPLEMENTATION OF SOFTWARE IDENTIFICATION PATTERNS

4.1. SOFTWARE IDENTIFICATION HINDRANCES

Effective SAM results in the ability to have accurate and complete view of software assets entitlements that are owned, deployed and used. However, if most of the recognition tools are quite efficient (especially in traditional architecture), a common mistake is to underestimate the process of identifying software after discovery. Indeed, most system admins can more or less easily compose scripts to collect program data or details on executable files; yet, the challenge is to associate this raw list of executables with normalized entitlements. There is a huge difference between software discovery, software recognition and software management. Here, the most common hindrances.

- File Header Information is composed by the titles and descriptions used to describe software when the manufacturer compiled it. It does not follow any industry conventions
- Add / Remove Program Data is well-known to be inaccurate and incomplete
- Normalization: Data needs to be normalized to rull out duplicates such as Oracle Limited, Oracle Corp and Oracle Inc.
- Suite Recognition – it is often not visible that a software instance recognized is part of a suite
- Footprints : some application have bundles or arrangements which may leave traces of installations – which at first look may look like a full installation e.g. a bundled version of SQL
- Recognition does not always allow knowing what is the version, if it is an upgrade, what is the level of services, professional, standard, personal, what is the language?

Most of SAM tools use software recognition databases and algorithms to scan raw files and provide information on what is installed. The aim is to find a description of software that is closer to what might be stated on the invoice when you bought it in order to perform reconciliation. Thus is does not eliminate most listed-above issues. Two crucial points are

- The possibility to make own modifications : for some in-house written software
- The possibilities to update this database each time new applications are developed. SAM tool vendors usually provide periodic updates or trickle down updates to download

Inventory has become much facilitated and in some instances free but the strength and intelligence of software recognition really varies and do not allow yet tools interoperability.

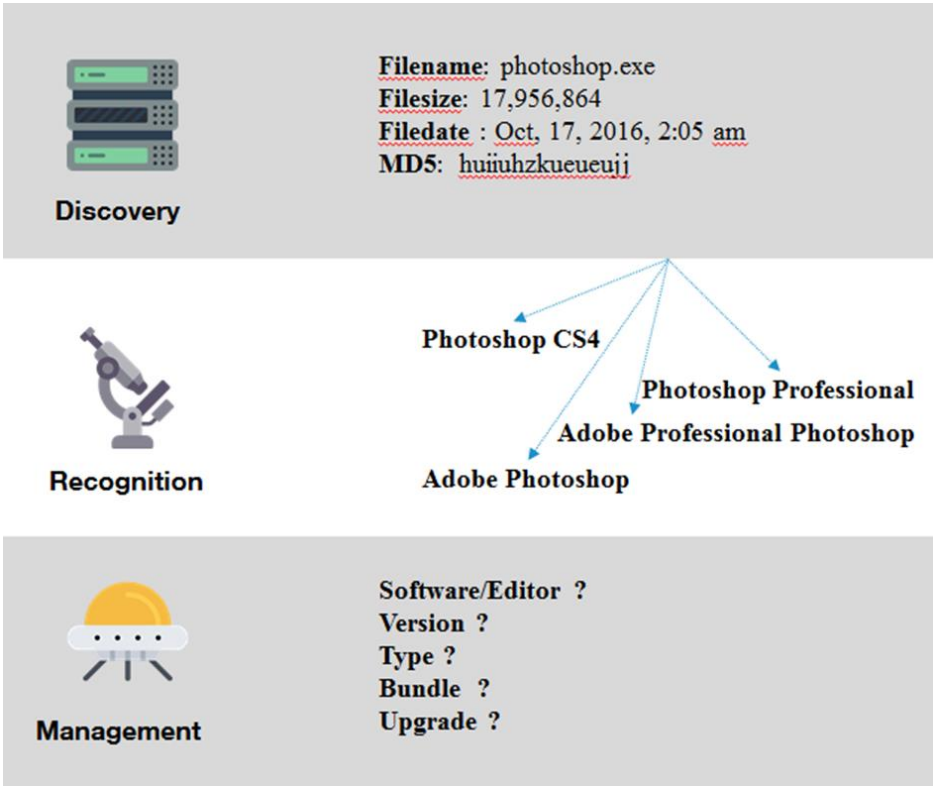

Figure 23 - Recognition vs Identification

The Figure 23 illustrates a real reconciliation using available tools. We took one product from one vendor (Photoshop CS4 from Adobe) and tried to perform a trustworthy reconciliation to demonstrate compliance. Software recognition saves a huge amount of time and frustration in manually crunching raw data and interpreting raw executable files or header information. Yet, different tools raise different recognition values, more or less relevant which do not allow identifying precisely software following requirements exposed in this section.

4.2. SOFTWARE IDENTIFICATION MODELS PROPOSED BY ISO 19770-X

Software identification tags (1) record unique information about an installed software application, including its name, edition, version, whether it's part of a bundle and more. SWID tags support software inventory and asset management initiatives. (2) Software entitlement tags will specify how license consumption measurement can be automated. This provides the next level of support for the automated software asset management process.

a. ISO 19770-2: about Software Identification Tags

ISO 19770-2 [27] was first introduced in 2009, with a recent revision released in 2015. SWID Tags are designed to help organizations identify what software are installed within their estate to help them verify their compliance position.

i. Purposes and scope

A SWID tag is added to a software package by the vendor before it is provided to the customer for deployment. It displays information about software, including name, edition, version, vendor and even whether it is part of a software bundle or not. It is up to the software vendor to populate the SWID tags with all of the mentioned information, and more, so that their customers can see what applications are in use.

As designed in ISO 19770-2, it is obligatory for the software vendor to provide SWID tags for their products and make sure that information provided is accurate. This level of accuracy is important, as without the right information in the SWID tag, it is not fit for purpose and can actually create problems calculating an effective license position for the vendor. The responsibility is with the vendor to adopt SWID tags and to make sure that each application has a unique identifier.

Because the SWID tag is created and populated by the software publisher in accordance with the ISO 19970-2 standard, SAM technology vendors were pressured to use SWID tags as the primary recognition where available. In theory, the SWID tags are infallible and a 'single source of truth' for the true nature of the installed application. These tags are used to normalize the installation data to help the SAM team identify what a software bundle or package is, without having to wade through incomprehensible .exe files or .msi packages.

A number of the world's leading software vendors support the SWID concept, such as Microsoft, HP and Symantec [28].

Key benefits associated with software identification tags inventoried in ISO/IEC 19770-2:2015 include the following:

- The ability to consistently and authoritatively identify software products that need to be managed for any purpose, such as for licensing, security, logistics, or for the specification of dependencies. Software identification tags provide the meta-data necessary to support more accurate identification than other software identification techniques.
- The ability to identify groups or suites of software products in the same way as individual software products, enabling entire groups or suites of software products to be managed with the same flexibility as individual products.
- The ability to automatically relate installed software with other information such as patch installations, configuration issues, or other vulnerabilities.
- Facilitate interoperability of software information between different software creators, different software platforms, different IT management tools, and within software creator organizations, as well as between SWID tag producers and SWID tag consumers.
- Facilitate automated approaches to license compliance, using information both from the software identification tag and from the software entitlement schema as specified in ISO/IEC 19770-3.
- Provide a comprehensive information structure of the structural footprint of products, for example the list of software components of files and system settings associated with a product to identify if files have been modified.
- Provide a comprehensive information structure that identifies different entities, including software creators, software licensors, packagers, distributors external to the software consumer, as well as various entities within the software consumer, associated with the installation and management of the product on an on-going basis.
- Through the optional use of digital signatures by organizations creating software identification tags, the ability to validate that information is authoritative and has not been maliciously tampered with.
- The opportunity for entities other than original software creators (e.g. independent providers or in-house personnel) to create software identification tags for legacy software, and for software from software creators who do not provide software identification tags themselves.

This part of ISO/IEC 19770 describes specifications for tagging software to optimize its identification and management establishes different roles like describe in Fig. 24.

- **Tag producers**: these organizations and/or tools create software identification (SWID) tags for use by others in the market. A tag producer may be part of the software creator organization, the software licensor organization, or be a third-party organization. These organizations and/or tools can broadly be broken down into the following categories.
    - *Platform providers*: entities responsible for the computer or hardware device and/or associated operating system, virtual environment, or application platform, on which software may be installed or run. Platform providers which support this part of ISO/IEC 19770 may additionally provide tag management capabilities at the level of the platform or operating system.
    - *Software providers*: entities that create, license, or distribute software. For example, software creators, independent software developers, consultants, and repackagers of previously manufactured software. Software creators may also be in-house software developers.
    - *Tag tool providers*: entities that provide tools to create software identification tags. For example, tools within development environments that generate software identification tags, or installation tools that may create tags on behalf of the installation process, and/or desktop management tools that may create tags for installed software that did not originally have a software identification tag.
- **Tag consumers**: these tools and/or organizations utilize information from SWID tags and are typically broken down into the following two major categories:
    - *software consumers*: entities that purchase, install, and/or otherwise consume software;
    - *IT discovery and processing tool providers*: entities that provide tools to collect, store, and process software identification tags. These tools may be targeted at a variety of different market segments, including software security, compliance, and logistics.

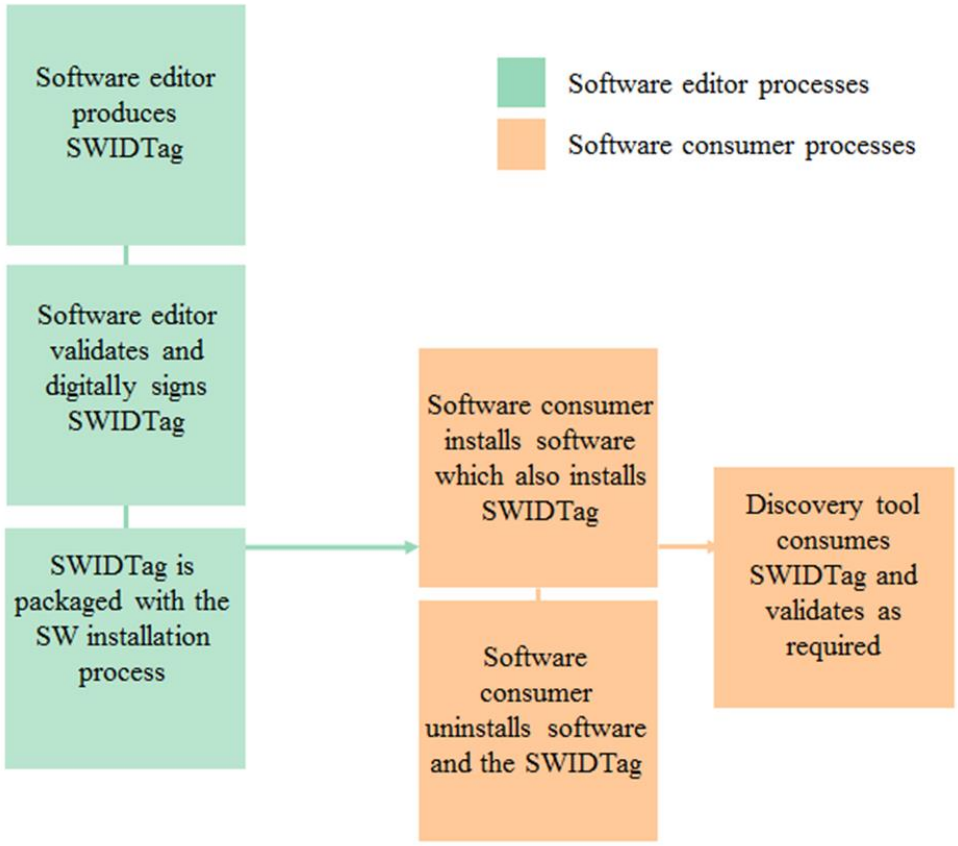

Figure 24 - SWID Tag lifecycle described in ISO 19770-2

ii. Implementation of SWIDTag processes

♦ **General requirements**

The software identification tag file shall be defined as an XML data structure. The XML schema definition (XSD) as specified in this revision may be found here (http://standards.iso.org/iso/19770/-2/2015/schema.xsd)

In instances where a software product is installed on a device, a software licensor conforming to this standard will ensure that a primary SWID tag is included on the installation media and installed at the same time the software is installed.

When software is uninstalled, or changed to a different release, the old SWID tags shall be removed from the device.

In instances where a patch is installed, the patch will include a patch SWID tag that will be installed when the patch is installed, and in most cases should be removed when either the patch is uninstalled, or when the product is

uninstalled, or changed to a different release. The determination if a patch tag is to be removed, or not, is based on additional data provided in the ownership attribute.

Supplemental tags provided by the software publisher (which may be used to identify relationships between software products) shall be managed in a manner similar to primary and patch SWID tags such that the supplemental tags should be removed from a device when the software product is uninstalled. SWID tags reside in the same directory tree as the applications installation directory tree. It is expected that if an application directory tree is deleted when an application is uninstalled, that the SWID tags associated with that application (including primary, supplemental, and patch tags) are deleted as well.

♦ **Elements**

Due to the multiple use cases identified for SWID tag creation, the minimum data requirements for a SWID tag are relatively sparse. The only values that are required for a SWID tag to be considered "valid" to meet the requirements of the XML schema shall be the following:

- Software Identity: represents the root element specifying data about a software component. A software product may be made up of one or multiple software components. Also, Software components may be atomic, or may be made up of multiple components. Each component will have its own SWID tag and only one SoftwareIdentity will exist for any one component.
    - Name
    - tagID
    - patch (default value is false)
    - Supplemental (default value is false ; If set to true, this tag specifies supplemental tag data that can be merged with primary tag data to create a complete record of the software information. Supplemental tags will often be provided at install time and may be provided by different entities (such as the tag consumer, or a Value Added Reseller).
    - tagVersion (default value is 0; The tagVersion indicates if a specific release of a software product has more than one tag that can represent that specific release. This may be the case if a software tag producer creates and releases an incorrect tag that they subsequently want to fix, but with no underlying changes to the product the SWID tag represents. This could happen if, for example, a patch is distributed that has a Link reference that does not cover all the various software releases it can patch. A

newer SWID tag for that patch can be generated and the tagVersion value incremented to indicate that the data is updated.)

- version (default value is 0)

These default values are specified so that if no value is included for these attributes, the SWID tag is considered to be the first version of a primary tag and that the software product has the version number of 0.0.

- Entity: Specifies the organizations related to the software component referenced by this SWID tag.
  - Role of TagCreator The relationship between this organization and this tag i.e.. tag, softwareCreator, licensor, tagCreator, etc.. Role may include any role value, but the pre-defined roles include the following: aggregator; distributor; licensor; softwareCreator; tagCreator. Other roles will be defined as the market uses the SWID tags.
  - Regid of TagCreator
  - Name of TagCreator

♦ **SWID supplemental attribute**

Supplemental tag data is data that is directly associated with a specific software product's primary tag but, for various reasons, the data included in the supplemental tag is not included in the primary SWID tag. SWID tag data may only be modified by the tag Creator; in other words, if a software creator provides a primary SWID tag for their product, the software consumer who installs and manages that software is not allowed to modify any data in the primary SWID tag. In this case, the software tag consumer can create a supplemental tag that provides specific details for the primary SWID tag they are referencing and they will set the attribute "supplemental" to the value of true. This supplemental tag can then be deployed with the installation of the software, or added after the fact as part of a device management process, or a software activation process. Supplemental tags may also be provided by the tag creator to add additional information related to a specific installation of a software entitlement.

♦ **Effectiveness of ISO 19770-2**

It is unrealistic to expect to create, manage, and use software identification tags without the use of automated capabilities built into specialist or generalist tools. Some facts about approach to SAM and SWIDtag are that companies do not have SAM tool that takes full advantage of the tags; instead they might slightly modify tool, or more often modify their existing processes; they create and deploy

tags manually. Likewise, observing current trends, it seems unrealistic that software editors will all adopt such generic approach described above and generalize adoption of Software Entitlement tags.

Despite the ISO 19770-2 standard first release date from 2007, only now are a few major editors (like Adobe, Microsoft, Symantec) starting to include SWID tags in their software packages (only in the new versions of software). As a result, a multitude of installed software across network will not have SWID Tags and therefore organizations will have to rely on usual software recognition methods. Moreover, SWID Tags are not unfailing:

- They still have to be created by humans which can make human mistakes.
- They do not address the problem of 'ghost' software on the network. A enduring challenge for SAM managers has been inventory solutions detecting fragments of software applications on devices, which are then 'recognized' as installed applications. In many cases, the application in question might have been removed from the computer. That is because it is common for files to be left behind after uninstall, or for files used by multiple applications to be detected and used to mistakenly assume that applications are installed.
- SWID tags can be guilty of creating 'false positives'. If the SWID tag is not removed as part of the uninstall process, that can lead many inventory solutions to report software installs that simply are not there.
- It appears that some flaw exists, like Adobe's one on Adobe Creative Cloud suite products: Individual products are given two SWID tags: One for the product, one for Creative cloud Suite. Discovery will show that the user has the full Creative Cloud suite installed, rather than the unique applications that are installed and being used.

SWIDTag have the potential to significantly improve the process of managing software and entitlements on condition that they are adopted accurately and unvaryingly by software editors. For now, they are not foolproof and not yet commonly used.

b. ISO 19770-3: about Software Entitlements Tags (Ents)

The ISO/IEC 19770-3 [29] standard for software entitlement tags is designed to integrate with ISO/IEC 19770-2, the standard for software identification tags.

The expectation is that software entitlement tags will not provide an interpretation of a software entitlement contract, but rather will specify how license consumption can be measured using automated means. This will be accomplished by providing:

- Metrics that must be collected from computing devices
- Measures against which, the metrics are compared
- Additional grants or limits placed on the entitlement

By providing specific details about what must be tracked in order to reconcile software entitlements, the expectation is that the SAM process can be automated and become much more accurate and useful to organizations, with a much lower administrative overhead.

Trustworthiness of Ents

This part of ISO/IEC 19770 does not require a specific process for generating content for entitlement files. Anyone or any organization may create Ents. The strong preference is for original Ents to be created by the software editors, so that these Ents have the highest degree of trustworthiness facing the licensing information they contain. However, there can be no assurance that all licensors/software publishers will produce Ents, firstly for new license transactions, and secondly for historical license transactions. Therefore, it should be possible for end-user organizations and third parties to create such Ents themselves. Furthermore, there are certain types of management transactions which would normally only be created by end-user organizations, but likewise these could also be produced by third parties depending on the circumstances.

Ents can never be assumed to have 100 % trustworthiness. Primary reliance should always be placed on normal contractual documentation, including invoices and terms and conditions for licenses which have been purchased or otherwise acquired. Given this warning, the trustworthiness of Ents is dependent on three things:

- *Authority*. Trustworthiness will depend on the authority of the person or organization creating the Ent, for the information given in that Ent. For example, the software licensor would be expected to have the highest level of authority for creating an Ent for a license it has sold, and therefore this type of information would have the highest degree of trustworthiness.
- *Authentication*. The information in an Ent needs to be authenticated to be certain of the level of trustworthiness which would be expected for

the Ent creator. The expectation is that Ents will be signed to provide such authentication, at a minimum for Ents which are created by one organization for use by another organization.

Universality. Models have to be general enough to encompass editors licensing variety, which is far from being trivial. Moreover Ents have to be effectively used by Software licensor and SAM tool. For now, only few marginal SAM approaches foreshadow Ents recognition and use; Not one of software editors have been announcing yet implementation of this part of ISO/IEC 19770-3.

5. WORKAROUND PROPOSITIONS

If ISO/IEC 19770 is currently the most advanced proposition to overcome software identification throughout it whole lifecycle, we underlined it relative efficiency in particular because of software market weak adherence (mostly for Entitlements)

For efficiency reasons, we propose to adapt the ISO/IEC 19770-2 with a concept borrowed from large retailers: Stock Keeping Unit (SKU)[14].

Originally, SKU represents warehousing item that is unique because of some characteristic (such as brand, size, model and color) and must be stored and accounted for separate from other items. Every SKU is assigned a unique not standardized identification number (inventory or stock number) which is often the same as the item's EAN (European Article Number) or UPC (Universal Product Code). For Software identification purposes, SKU identify Software and its PURs. To be informative, here are two vivid examples.

5.1. ABOUT PUR

Purchasing a train ticket. For the same journey, a myriad of options and variations and the price can vary significantly. Among others:

- type of ticket (flexible or no),
- time of the day (peak or off-peak)
- class (first or cattle)
- age of customer (infant/child/adult)

[14] http://www.businessdictionary.com/definition/stock-keeping-unit-SKU.html

- special programs (season ticket, student card, loyalty card).

Like illustrate on Fig.25, it is the same for the software industry, licensing provides options and flexibility, called PUR. PUR are (not exhaustive) : customer's use rights, Rights to use other versions, Applicable Use rights, disaster recovery rights, permitted periods of use, conditions on use, required used of some product, metric.

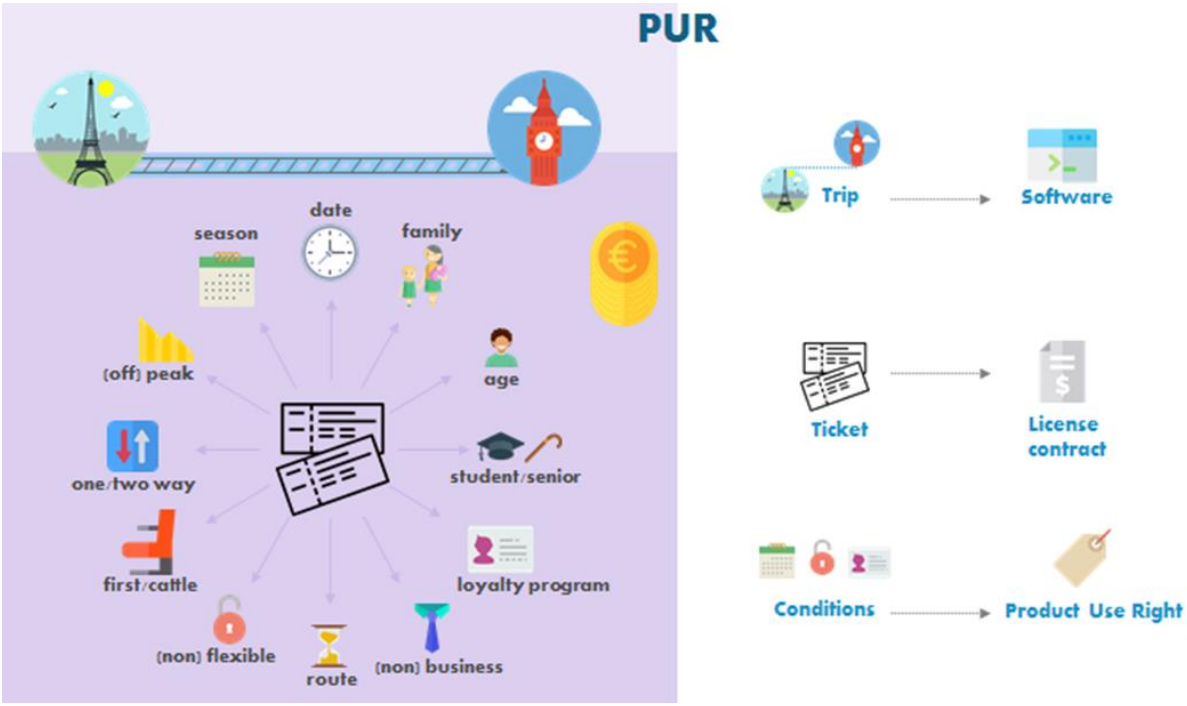

Figure 25 - Analogy between PUR and railway

5.2. About SKU

On the shop's juice shelf, on Fig.26, the same orange juice from the same producer can be sold in three different packaging: containers like a glass bottle, a can and plastic bottle. These three products containing the same juice will have three different SKU. But if we put three glass bottle of this juice in our basket, they will have the same SKU; it is not possible to find any difference between them. Making a parallel between Software and Juice: Software is the content (Juice), and Product Usage Rights are the packaging (PUR) (Bottle).

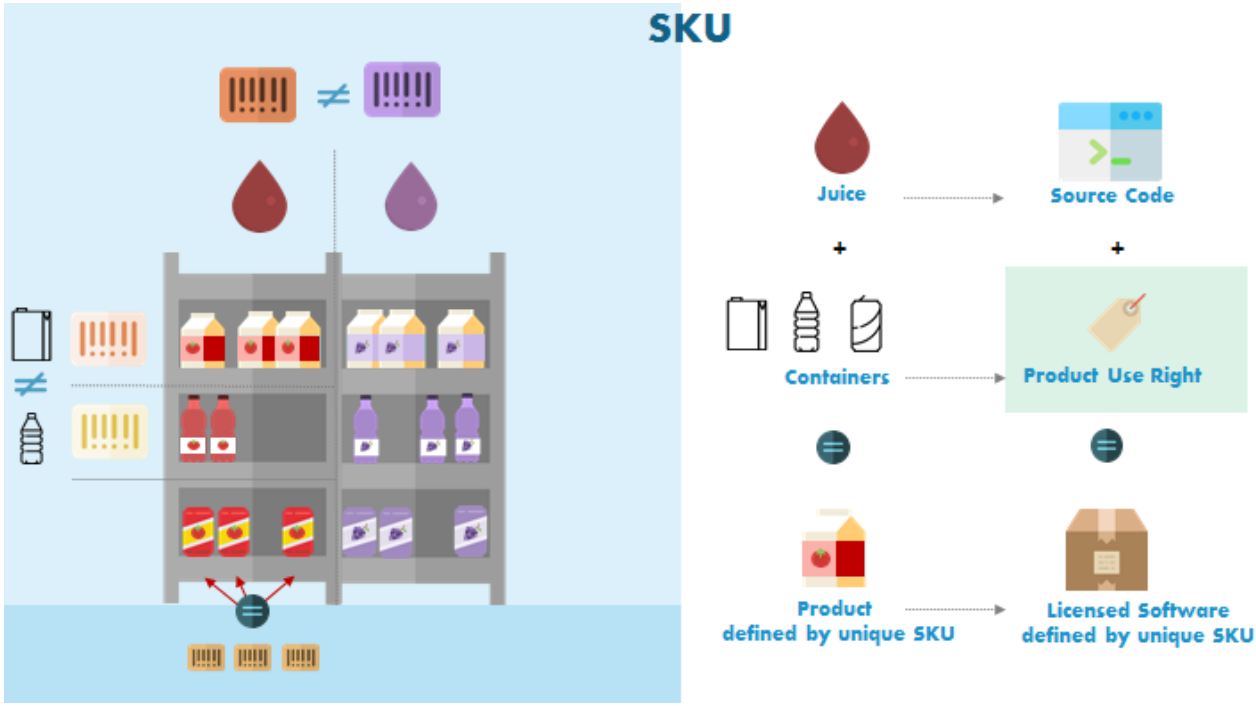

Figure 26 - Analogy between SKU and Juice Bottles

SKU is a not normalized code defined by the Software editor. It can be found very often in product catalogues/purchase orders during purchasing phase. Our experience showed that this code is internally used by software editors, software licensors and software retailers for stock management and orders management to the detriment of usual software identifiers like name or commercial denomination. It guarantees higher accuracy and reliance. Strangely, this concept is not widely spread for SAM purposes: identification, usually alphanumeric, of a particular product that allows it to be tracked for inventory and software entitlement purposes.

The term “stock keeping unit” is traditionally associated with physical goods. In the sense of licenses it refers to a unique identifier, sometimes also called “part number». The term “stock keeping unit” is typically associated with unique products for sales purposes, such as software entitlements. It may not correspond uniquely to specific software products, but may instead represent packages of software, and/or specific terms and conditions related to software products, such as whether it relates to a full product, upgrade product, or maintenance on an existing product. SKU allows to identify precisely requirement exposed in III.2.2 (deployment and migration conditions, access and usages, geographical and location restrictions and elasticity scope).

5.3. PROPOSED IDENTIFICATION LIFECYCLE

Combining these two notions by including SKU in the SWIDTag allows identifying with the highest accuracy Software and it PUR. The right approach has to be to combine the ability to read SWID tags with a sophisticated software PUR recognition methodology which has the capability to challenge the information held in a SWID tag (to prevent false positives) and provide accurate software entitlement recognition even in the absence of a SWID tag (Fig. 27).

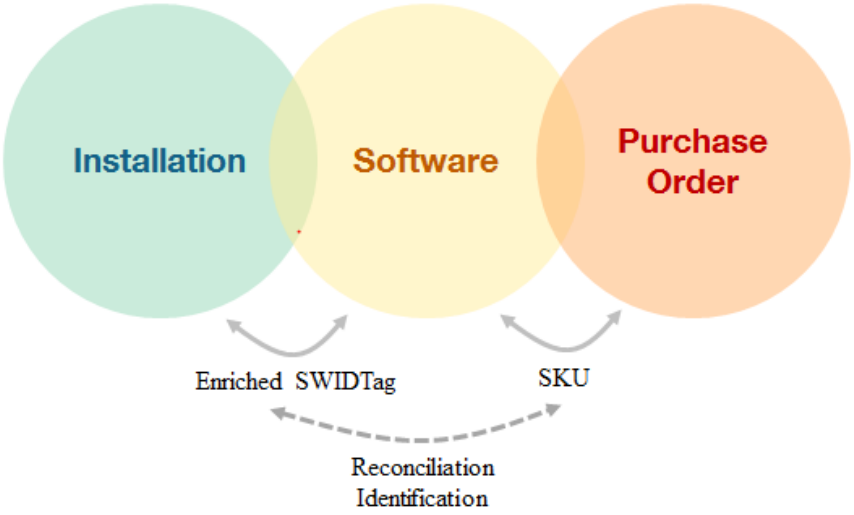

Figure 27 - Combination of SWID and SKU

In Fig. 28 and 29, we propose to present software identification lifecycle using enriched SWIDTag (SWIDTag +) where SKU will be intercept from purchasing phase and included in SWID supplemental attributes. A SKU database should be created and maintained to "translate' the codes into understandable data which can be integrated by SAM tools in order to match deployment, access and PUR.

Fig 28 represents SWIDTag+ lifecycle when the software editor is providing a SWIDTag.

Fig 29 represents SWIDTag+ lifecycle when the software editor is not providing SWIDTag.

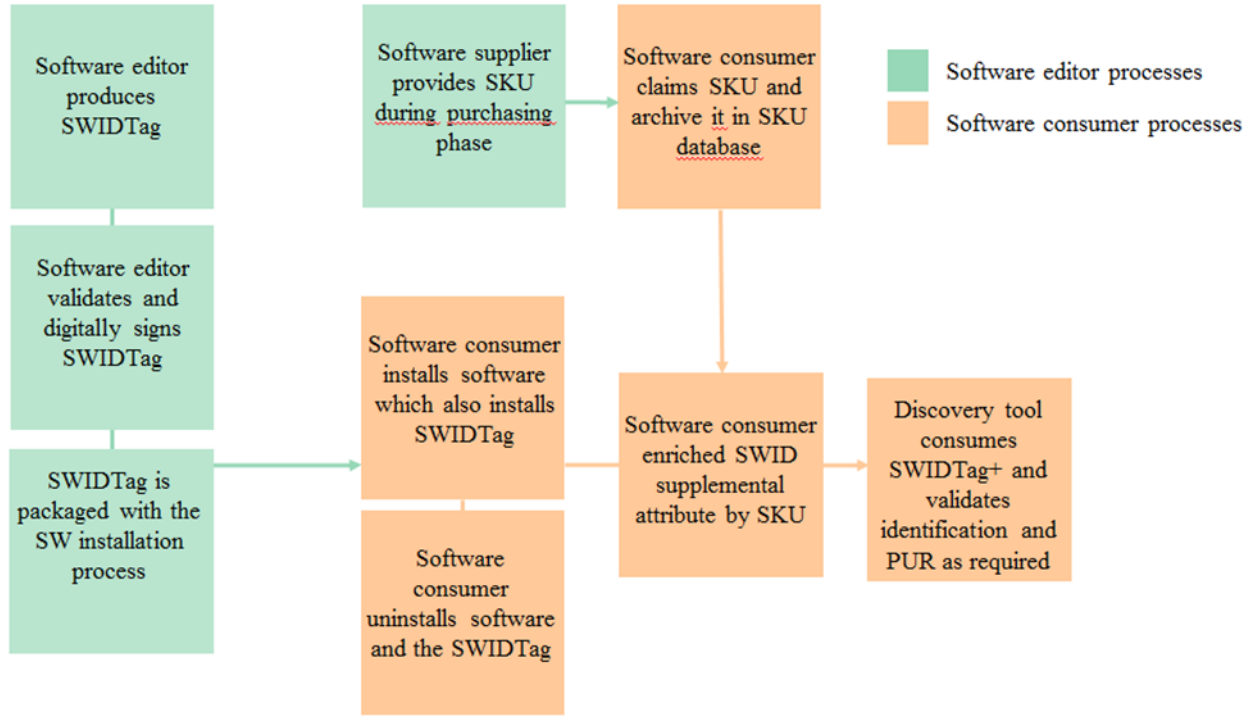

Figure 28 - SWIDTag+ lifecycle with initial SWIDTag

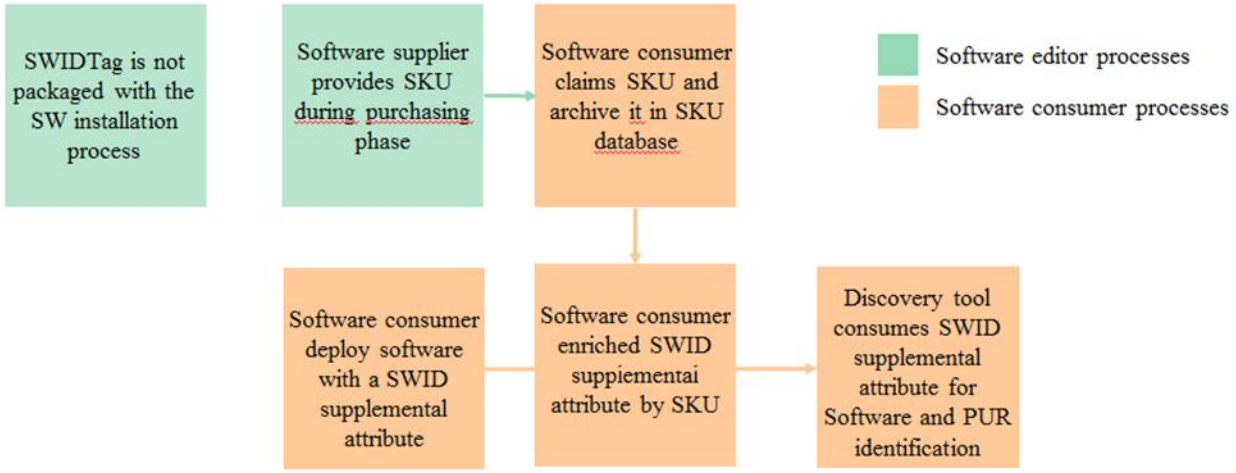

Figure 29 - SWIDTag+ lifecycle without initial SWIDTag

In Fig 30, we propose to link the identification concerns with the purpose of next chapter: how to identify a software package through the whole software lifecycle to be able to implement SAM controls of compliance and optimization. In this figure we will refer to the concepts proposed in Fig. 22:

(1) The software suppliers develops software packaged enriched by a SWID tag which will be proposed under different licensing models (from 1 to n). A prerequisite is that the suppliers will also give clear access to a service catalog composed by transparent explanation of licensing conditions, restrictions, prices and metrics using SKU as unique identifier of each couple software/licensing models.

(2) During procurement & delivery phase, the software customer is choosing software and a specific licensing model and enriching the software package with corresponding SKU (in supplemental SWID Tag previously described).

(3) Software packaged is deployed in cloud environment and each instance be identified using SWID Tag +.

(4) Software can be combined to create applications which might be proposed to final customers. Applications are tagged by the service provider and to allow identification of each software component, related entitlements, identification of application itself and specific sub-licensing conditions proposed by the service provider.

(5) SAM tools have access to procurement, instantiation and usage identification's data on software and application levels and allows (6) charging and billing toward software suppliers and eventual final customer.

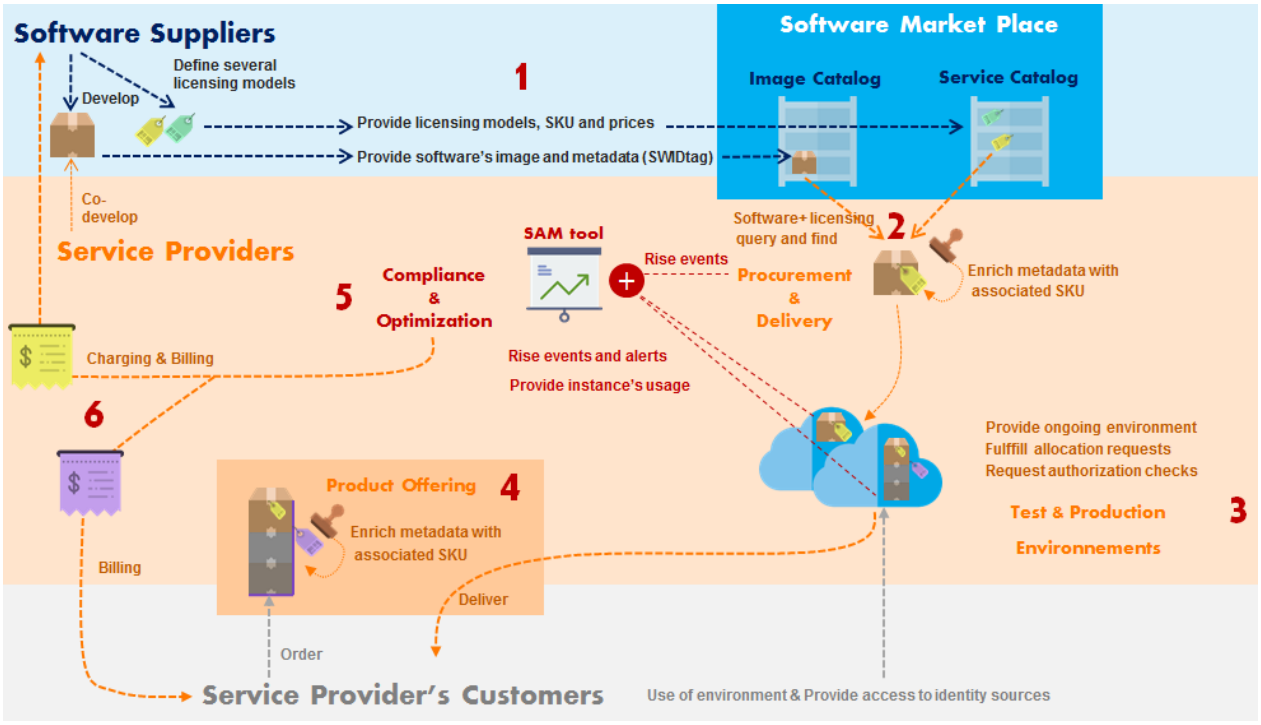

Figure 30 - Software identification lifecycle from provisioning to billing

Chapter 4

IV. Proposition of a SAM Model for the Cloud

Proposition of a SAM Model for the Cloud 105

1. SAM Control loop 106

A. Autonomic computing and general concept of control loop 106

B. Application to SAM Control Loop 107

C. SAM model, sensors and effectors 110

01. Software abstraction layers 110

02. Cloud resources Identification 111

03. Software lifecycle 112

04. Dynamic reconciliation 116

i. Complete discoveries and inventory consolidation 117

ii. Dynamic software recognition 117

iii. License understanding and compliance 117

iv. Dealing with use-scaling 118

v. Focus on Usage Collection 118

2. Database model for SAM loop 121

A. Relational Databases vs Graph Databases 121

B. From relation to Graph Databases 122

C. SAM Graph Proposition 125

Software Asset Management is not only improvement of license compliance or cost-cutting, it is mainly about deciding about a strategic approach of understanding software needs so that their deployment's efficiency and effectiveness will contribute to maximize the return on investment. The fact is that license optimization requires a major shift within a company to implement proactive SAM processes and be able to harness the power of this decisive business asset. We propose (1) to develop our propositions for optimized SAM model and processes and usage collection cases inspired from our experience in Orange SA, (2) and to discuss about a graph database as a central process data connection.

1. SAM Control loop

1.1. Autonomic computing and general concept of control loop

Organizations need to reduce their software costs, simplify the management of complex software licensing, and ensure the highest possible levels of system availability, performance, security and asset utilization. Autonomic Computing addresses these issues through a fundamental, evolutionary shift in the way that IT systems are managed.

Autonomic computing is about shifting the burden of managing systems from people to technologies. (IBM, 2005)[30] proposed a high-level architectural blueprint to assist in delivering autonomic computing in phases. The architecture reinforces that self-management uses intelligent control loop implementations to monitor, analyze, plan and execute, leveraging knowledge of the environment. These control loops can be embedded in resource run-time environments or delivered in management tools. Autonomic managers and manual managers communicate with managed resources through the manageability interface, in the form of a touchpoint, using sensor and effector interfaces. A sensor interface exhibits two interaction styles, the retrieve-state interaction style (used to query information from a managed resource) and the receive-notification interaction style (used to send asynchronous event information from a managed resource). The effector interface exhibits two interaction styles, the perform-operation interaction style (used to set state data in the managed resource) and the call-out request interaction style (used by a managed resource to obtain services from some other external entity in the system).

Basic concepts that apply in Autonomic Systems are closed control loops. Essentially, a closed control loop in a self-managing system monitors some

resource (software or hardware component) and autonomously tries to keep its parameters within a desired range.

1.2. APPLICATION TO SAM CONTROL LOOP

Basically, SAM aims to manage two types of risks: Counterfeiting which represents any default in license compliancy (Fig. 31), and over-deployment (Fig. 32) which consist in more deployment than measured needs.

The model's fundamental functions (further developed in next section) consist in:

- Elaborating a consolidated software view based on contractual, deployment and usage facets. It implies to comprehend a collection of heterogeneous data and organize the optimal state of SAM processes
- Allowing software lifecycle accurate identification
- Handling and interpreting several licensing rules
- Anticipating, diagnosing and react to counterfeiting
- Discovering, diagnosing and react to over-deployment
- Comparing software usages and simulating licensing model's changes
- Identifying the best licensing model's according to current and forecasted software usages

The decision part can be described as Event Condition Action (ECA) rules[15]. An ECA rule has three parts: an event, a condition, and an action. The semantics of an ECA rule are: when the event has been detected, evaluate the condition, and if the condition is satisfied, execute the action. (Tab.4) shows non-exhaustive list of our model rules.

Event	**Condition**	**Action**
A new software instance is detected	The instance and associated PUR are properly identified	Link an instance and a stock

[15] ECA rules are used within active databases for supporting reactive behavior and were first proposed in the HiPAC project: Dayal U., Blaustein B., Buchmann A., et al. S.C. HiPAC: a research project in active, time-constrained database management. Tech. Rep. CCA-88-02, Xerox Advanced Information Technology. Cambridge, MA, USA, 1988.

License stocks changed	The software product is properly identified	Link stock and with bound instances
New link between instance and stock	Measure instances usages and analyze deployment conditions	Validate compliance
New licensing business model	All impacted objects exists in the data model	Implement new metric rule

Table 4 - Some ECA model rules

Sensor interfaces consists in interacting with:

- License stocks: more specifically, consists in intercepting all software license sales and purchases. It encompasses knowledges on software and its PUR identification
- Deployments: this consists in intercepting each software instantiation and allowing queries about bound workflow configuration
- Usage: this consists in intercepting different usage metrics and allowing queries to workflow consumed resources

Effector interfaces exhibits several interactions:

- Operation on license stocks: which consists in increasing license stocks (purchasing new licensing, migrating stocks from another entity), decreasing license by internal or external transfer. It can be contract renegotiation or third-party supplier changes.
- Deployment changes, it consists in changing workflow configuration to change resources allocation or consumption, migration, software de-installation
- Usage: consists in setting/updating/removing access controls

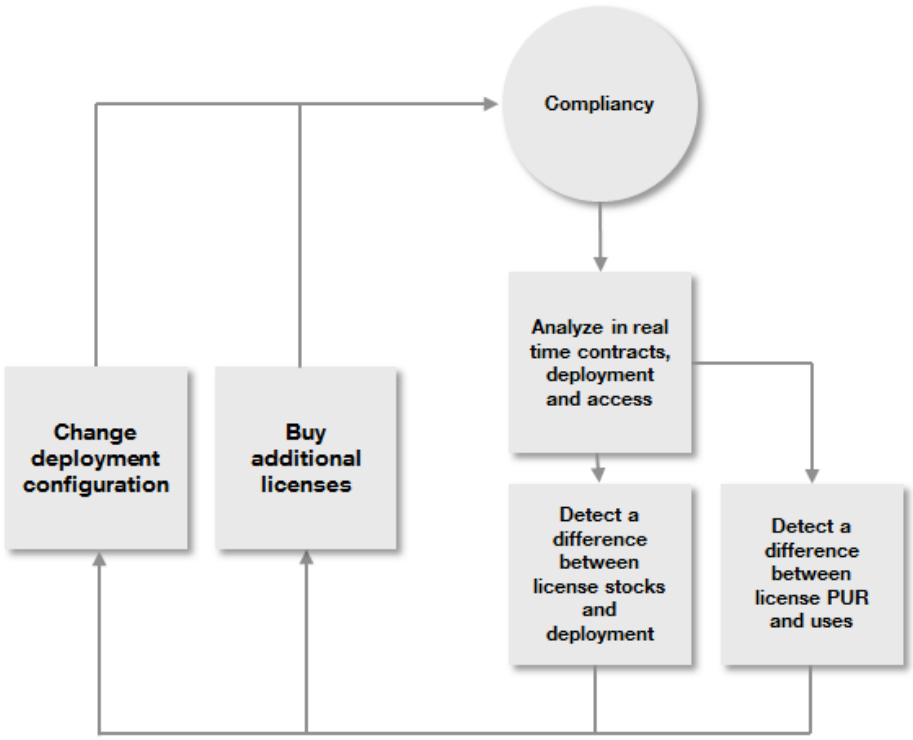

Figure 31 - Compliancy control loop

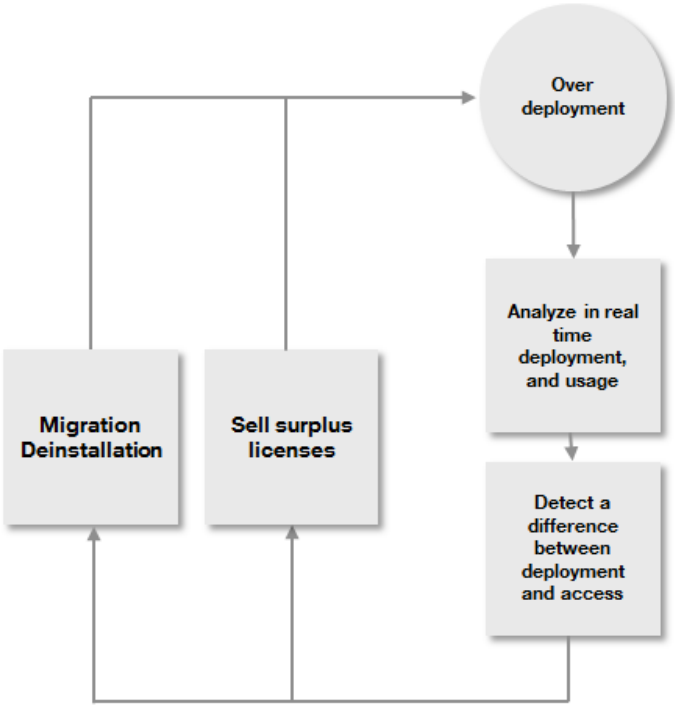

Figure 32 - Over-deployment control loop

Our SAM proposal takes into account the complete software lifecycle, considering that each step feeds a SoftWare DataBase (SWDB) and that every step is accompanied by one or more SAM control (considered as sensors). All possible information related with the use of software should be captured and stored in order to implement all the required usage controls.

Through those controls, the SAM processes analyze the current situation in real-time, confront the use of services with the license stock. SAM processes also take potential optimization decisions, like described previously in control loops. In Fig. 33, we name each step of the software lifecycle, the dynamic adjustments and reconciliation necessary to introduce the next section.

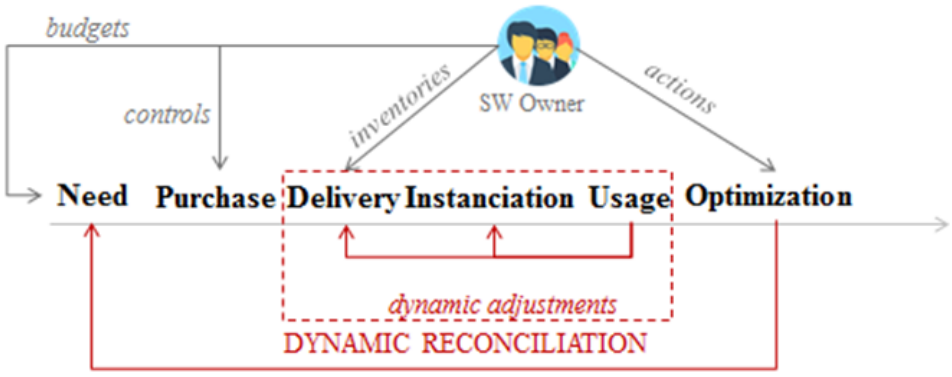

Figure 33 - SAM lifecycle

1.3. SAM MODEL

Accurate SAM model should allow representing platforms and software abstraction layers, deployments and resources in order to evaluate related costs and compliance risks given any licensing model from the simplest to the most intricate. We assume here the necessity to collect cloud resource information on each software abstraction layer. It will enrich a series of SAM processes described as the software lifecycle. The characteristics of software, its instances and their consumption have to be traceable and confronted to acquired rights and costs, by means of these lifecycle elaboration processes.

a. Software abstraction layers

To illustrate the abstraction layers of a software system, we refer to S.Kachele (2013)[31] Cloud taxonomy for Computation, storage and Networking. He discussed the relevant abstractions bottom-up. Effective SAM necessitates having a

view on the cloud resources monitored on each abstraction layers presented. Yet, access to these data is not always possible depending the layer (IaaS/PaaS):

Hardware (HW) represents the least abstract layer in his hierarchy. It provides bare metal resources such as CPUs, computing cores, the amount and type of RAM, co-processors and other hardware devices like network interface cards and storage controllers.

Operating systems (OS) reside on top of physical or virtual hardware. They provide isolation features resulting in the ability to execute multiple processes from multiple users and to share resources. From an operating system point of view any application being executed is a process. Yet, with respect to abstraction and programming, multiple types of applications exist. They may directly make use of OS functionality or apply further software components with a possibly different abstraction.

A runtime environment (RE) creates a container for the execution of applications. The OS provides a basic runtime for all processes running in a system. Higher level runtime environments might significantly enrich the OS runtime and further intermediate runtime environments. Typical features of that layer are the execution and interpretation of intermediate code and sophisticated libraries that applications can use.

The framework layer (FW) uses mechanisms provided by RE and OS.

The application (APP) contains the business logic. It is located on top of the software stack and can be run on any layer above HW. Yet, it is commonly deployed on RE or FW layer. An APP may or may not be accessible for clients.

b. Cloud resources Identification

The characteristics of software, its instances and their configuration/consumption have to be traceable regardless of the deployment conditions. This requirement of traceability encompasses precise identification of Software and resources to allow maintenance of deployment inventories, usage-based reporting of software deployments, internal elasticity management inventory to fulfill entitlements requirements.

IaaS offers bare HW resources to a tenant. As access is granted at a low level the tenant is free to install and configure arbitrary software. Yet, this means that the tenant is fully responsible for running and managing the entire software stack. The provider has to maintain the hardware. He may support scalability by offering mechanisms to spawn new machine instances and thus extend the resource pool of

the tenant. Yet, the tenant remains responsible for its application exploiting the larger pool.

The PaaS allows tenants to deploy applications in a cloud environment. In contrast to IaaS, it no longer provides the perception of a computing node to the tenant any more. At most, the tenant may observe multiple instances of his application. PaaS represents the highest layer that still allows tenants to deploy application logic. Tenants receive a fully managed software platform. Yet, it is provided as a framework so that applications remain passive and are invoked by the cloud controller. The provider has to maintain the environment and to manage the underlying infrastructure including FW implementation. We need to implement or strengthen bridges to recreate computing node perception and have transparent view on the entire software stack.

c. Software lifecycle

In its basic form, the software lifecycle that we consider is composed of 5 + 1 steps as shown by (Fig. 34 - 37); each step corresponds to at least one process. Some process can be played several times. We underline the necessity of strong cooperation between the departments in charge of each part of the lifecycle; it includes among others procurement, operations, IT and controlling. All the processes are necessary to build the software lifecycle. Its accuracy is a catalyst of compliance and optimization guarantee.

♦ **Needs**

In this process, the consumer justifies his need and choice of software. Allowing employees to make ad hoc purchases and forget about controlling authorized purchases is a common mistake. Companies often buy licenses as needed bit by bit, rather than under a volume agreement, which can be much more cost effective. Need should be also confront to internal available stocks, through second-hand license market, supposing that there is no available substitutable products.

♦ **Purchasing**

This step encompasses sourcing processes, negotiation, contract, billing etc. We underline the necessity to have a central repository to keep proves of purchased licenses and conditions. Easy access to this data allows fast-checking in case of editor audit requests or internal compliancy audit. We underline the necessity to keep tracks of software license agreements and renewal dates, makes enterprises vulnerable to lapses in Software assurance or other maintenance programs.

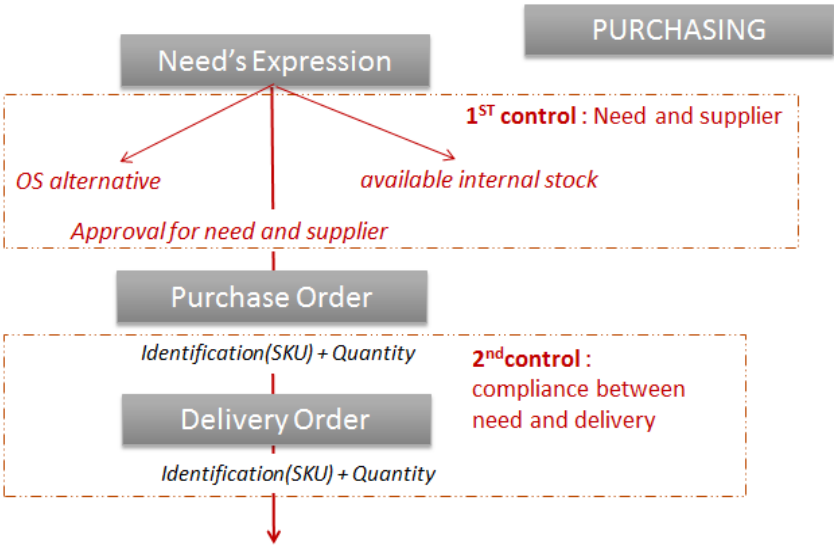

Figure 34 - Software Lifecycle - Need & Purchase

♦ ***Delivery***

This process encompasses the software receipt via downloading platforms, preparation for installation on user platform, entry into a software catalogue. We underline the critical role of delivery to guarantee the respect of the Product Use Rights (PUR) defining how software licenses can be consumed. They include (among others) upgrade, downgrade, second use, virtual machine use and multiple version rights. They are typically specified in the license agreements. Product Use Rights can vary from product to product and version to version. Accurately respecting PUR can significantly reduce risks related with counterfeiting and over-deployments.

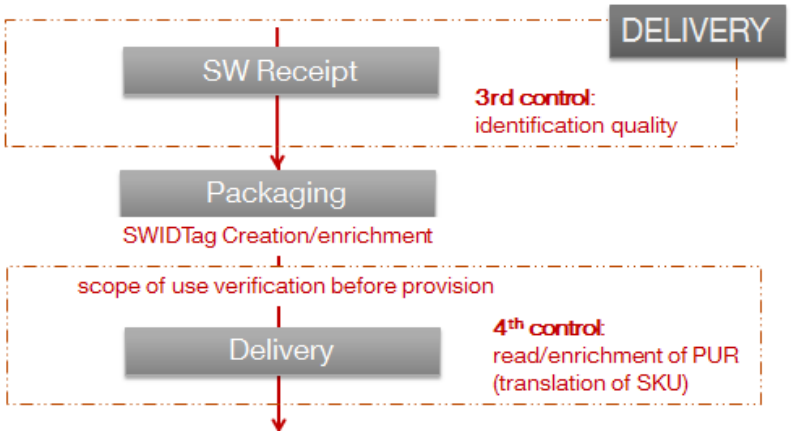

Figure 35 - Software Lifecycle – Delivery

♦ **Instantiation**

In this process, software is installed in an environment (for instance, a given Cloud), in other words, software is able to be used. By tracking software installations/instantiations, a company can be able to significantly reduce risks – either because the applications might not be in use (risk of over-deployment) or because of piracy (either accidental if PUR are not respected or intentional). We will develop this idea further down in this section.

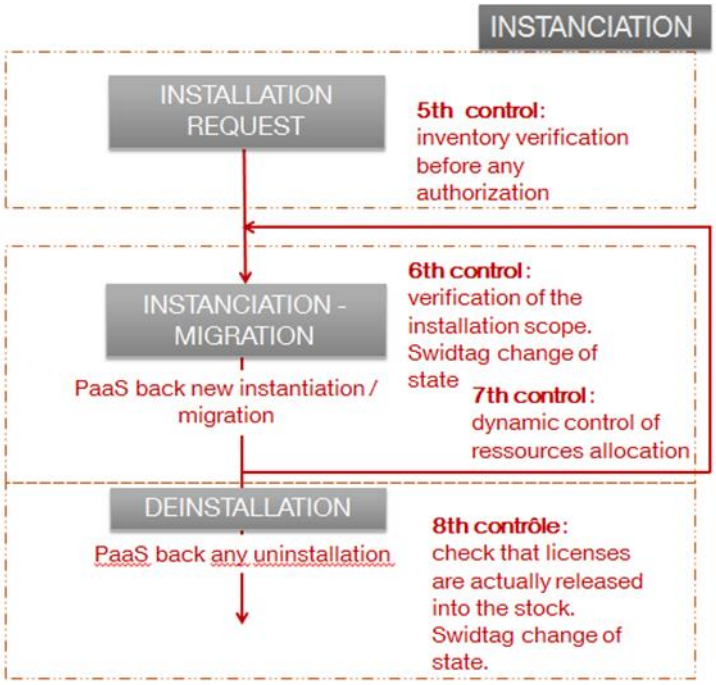

Figure 36 - Software Lifecycle – Instantiation

♦ **Usage**

A user consumes a service/software. In this process, we have to identify the cases where multiple users consume the same service simultaneously and translate this in terms of use (multiplexing, multidevice …). By tracking software usage, a company can be able to significantly reduce risks – either because the applications might not be in use (risk of over-deployment) or because of piracy (either accidental if PUR are not respected or intentional).

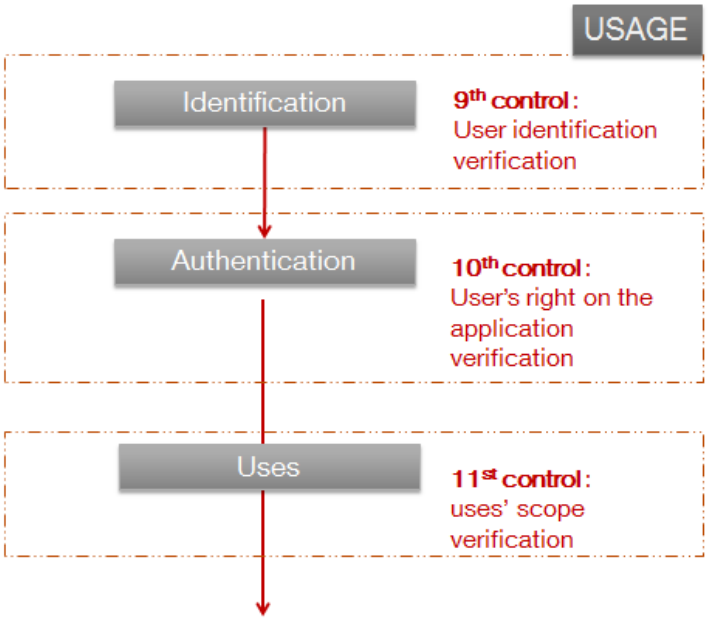

Figure 37 - Software Lifecycle – Usage

♦ **Optimization**

This process corresponds to confronting the need/contract/installation/use with the license stock according to a measure of consumption previously defined (metric). Here we can create a model of costs for any measure of use and identify the most suitable scenario of consumption or of customer billing. Optimization also means that companies have to plan and schedule software needs and purchases on longer term, based on use observations and predictions. Ordering licenses without determining what the company truly requires on the horizon could be an expensive mistake. Likewise, optimization is keeping track of software license agreements and renewal dates to be able to purchase or renew/not renew maintenance in the right moment, depending observed and predicted uses or according to strategic plan when new release of software are expected (actually, if you buy maintenance before announcement of a new release, the price might be lower but it will be eligible for that product upgrade.

Guarantee that SAM processes are in place to provide the necessary use intelligence and make extra licenses available. Removing unused instances ensures cost-optimization, as freed-up licenses can be pooled for future use. This removes overspend on licenses and it costs are reduced through automated adjustments.

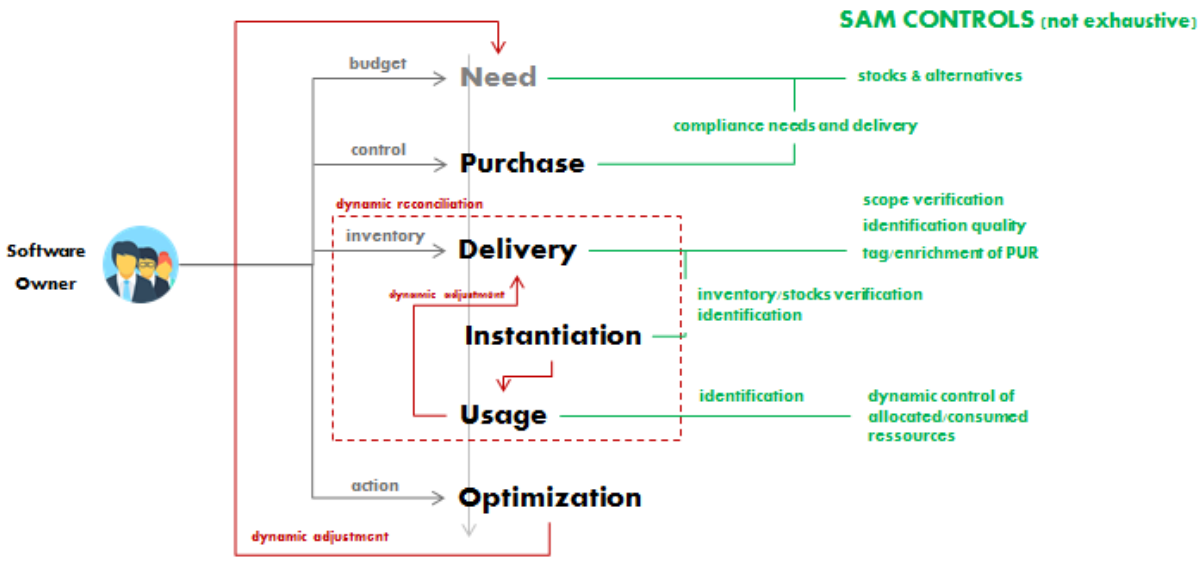

Figure 38 - SAM general retroaction loop and control

To operate dynamic adjustment shown in Fig 38 and operate sensors and effectors, we need to have a clear view on each steps of this lifecycle, meaning an exhaustive view on purchased license stocks and bound entitlements, precise software allocation and consumption and it corresponding workflow resources.

d. Dynamic reconciliation

For whatever reason, the licensing rules frequently change and companies need to stay on top of all the vendor rules and regulations. The software lifecycle is not only in one-dimensional. It must be considered as a loop from needs to uses. Not thinking so can result in non-compliancy. Virtualization and cloud computing intensify this situation by increasing dynamicity of environments, where rights and restrictions must be in real time investigated to ensure license compliancy. For usage-based license, real-time management and monitoring is even more inescapable and we should not underestimate the complexity of tracking software's license compliance and uses. However difficult, an optimized license program has to be set up, as automated as possible. Software license optimization tool, also known as next generation SAM tool allows to collect and confront all the necessary data – from procurement stage to technical inventories – and to rightsize software estate to reduce risk, ensure compliance and save significant costs. One of the ways to create cost saving lies in effective Software license optimization controls - and tools to support it.

Part of our experience in Orange reveals that organizations have some level of under-licensing on their virtualized network (using more software than what they purchased) and they fail to rightsize their usages of licenses they have bought. Mainly, there are no continuous processes setting up a performant SAM approach like proposed in figure above and allowing dynamic adjustments. These processes encompass the followings.

i. Complete discoveries and inventory consolidation

It is crucial to know what devices and software are present on the corporate network, across each and all major platforms. Ideally, discovery or inventory tools should track software use on every device from mobile to computer through datacenter, servers and into the cloud in order to redeployed redundant software to other users.

Most of the time, organizations need to use more than one inventory technologies to audit the entire multi-platform network. If so, it is crucial to merge the disparate inventories together into a single asset repository with unified naming conventions.

ii. Dynamic software recognition

One of the biggest issue if to associate the known software licenses with what is truly installed on machines across the network. Manually identifying software is not conceivable because this process is too slow and leading to many mistakes. It is necessary to rely on a highly-effect software recognition process to recognize commercially-licensable software.

iii. License understanding and compliance

The continual introduction of new license models, often promoted to be 'simplified', mainly adds complexity to manage SAM within an organization. At the most, new licensing models just mean more to manage (actually, vendors rarely retire older licensing models at the same time as introducing new ones).

Imagining the worst scenario, the new licensing schemes are themselves more complicated than previous ones. The move towards use-based license models increases both the number of metrics that need to be managed and the complexity of the actual calculation methods for assessing the appropriate licensing for different scenarios. Indeed, SAM has to deal with different notions like 'named user', 'processor' and capacity-based licensing models or 'data traffic consumption' etc. (in annexes, some examples of different metrics and their calculations). It multiplies the controls and usage capture tools.

Very often, these metrics are too difficult to manage manually, especially that we have to deal with many vendors' distinct licensing models. Effective SAM needs to integrate as fast as possible all types of new software license models and translate it into automatic intelligence to calculate products usages right controls and implement it – including upgrade and downgrade rights which are essential to optimized license management.

Then, another lock is the ability to produce understandable reports based on usage and resource consumption. SAM must feed each stakeholder from across the organization and enable them to view, generate and interpret compliance and management reports through an intuitive multi- user, roles-based interface. We have observed in Orange that each stakeholder's involvement and awareness is a sine qua none condition to get valuable result for a SAM program – from CIO to user, passing through operational.

iv. Dealing with use-scaling

Nowadays, users access data and consume software on multiple devices, from wherever they are. We must integrate this scale effect thinking about software license optimization and particularly focus on the multi-dimension of Software lifecycle. If a company cannot understand the entire view of the software estate, both on physical and virtual machines/containers (and indeed understanding the relationship between virtual guests and their physical hosts), then it is impossible to calculate and prove that the right licensing models are being used. Usage-based licensing force to consider a new dimension of SAM issues: the market is moving toward a preference for the subscription license, largely due to the cash flow benefits, inclusion of value-added features and services, and flexibility of a pay-as-you-need model. For the SAM' point of view, real-time consumption models more and more often proposed, questions the validity and relevance of selected criteria. We propose to examine this point below.

v. Focus on Usage Collection

Fig. 39 proposes to differentiate usage in three categories: allocation – supervision - consumption. Each variation might represent a metric (in that licensing meaning). Allocation covers resource configuration like virtual machine (VM) host, maximum allocated VM resources. It represents theoretical resource uses unlike consumption which encompass real resource uses, observed traffic, consumption of service, object, time, access. Supervision is not based on resources allocation or consumption but on the service ability to manage/create objects or services. Typically, an orchestrator use can be quantified by its amount of managed/created container. This usage distinction allows to link usages and

licensing models and to forge a bond between the software licensing costs and service providers' business value-added.

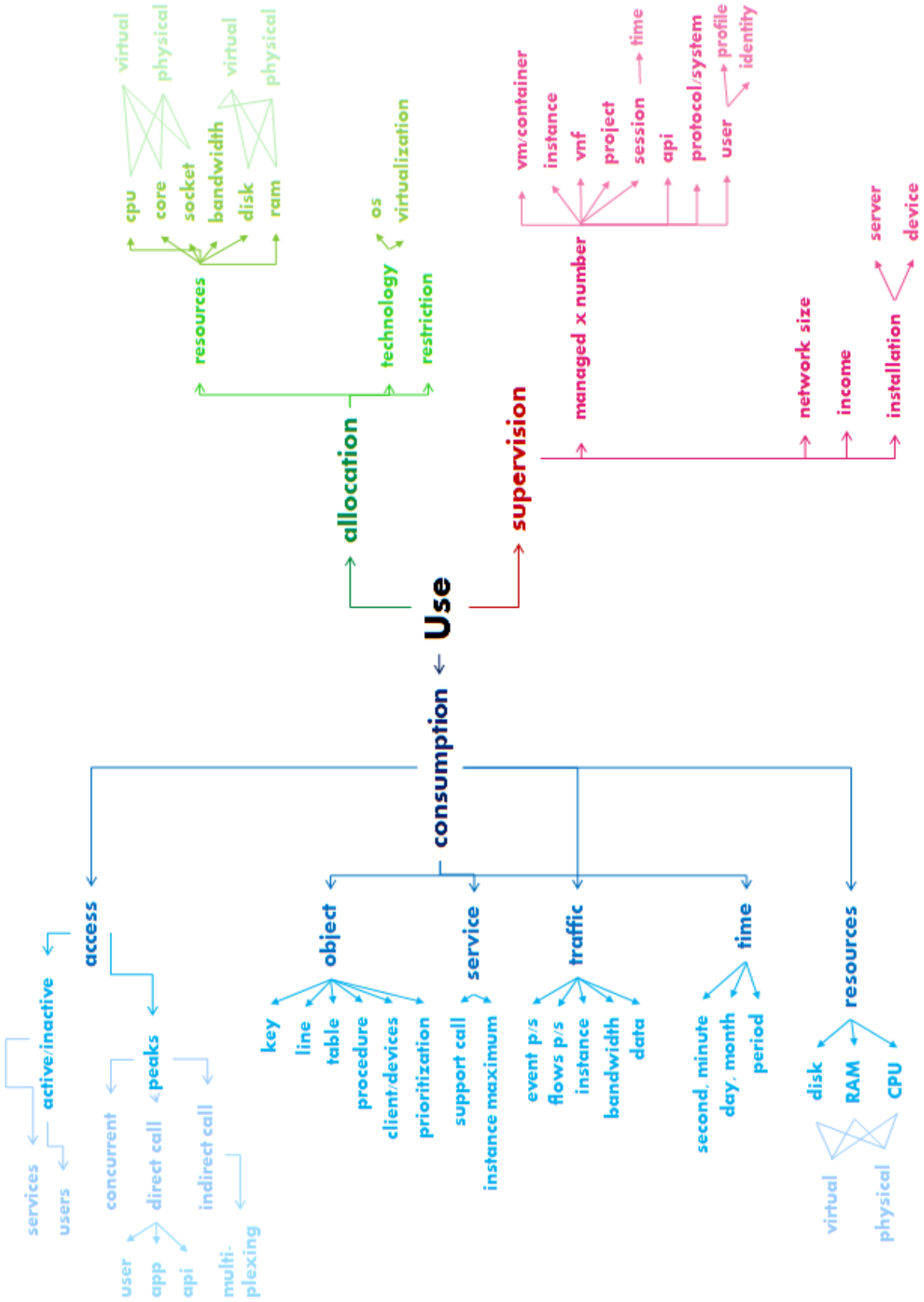

Figure 39 - Different measures of uses translated into licensing models

2. DATABASE MODEL FOR SAM LOOP

Access to information is a key for effective sensor and effector operation. The way information is stored has direct influence on the ability to use it. SAM distinctive characteristic lies in heterogeneity of managed resources (contracts, usages, deployments) and strong relation impacting them. Technical environments which have to be monitor are vast and necessitate plurality of sensors, just like licensing possibilities. It involves the highest flexibility in data model. For this reason, we propose to discuss why graph database should be used to design SAM loop. There are several arguments:

- To recognize relational structures
- To identify relations and work directly with entities by relational groups
- To benefit from an alive system (versus legacy storage system) by exempting from some technical limitations (joins, foreign-keys…)
- Anticipation of constantly changing the orientation of the system (and thus greater flexibility)
- For constant improvement of features

2.1. RELATIONAL DATABASES VS GRAPH DATABASES

Relational databases have been the standard of software applications since the 80s, and nothing really changed so to this day. These databases are able to store highly structured data in tables with predetermined columns of certain types and many rows of the same type of information. This organizational rigidity requires to formally structuring the data in the development of applications and storage.

References to other rows and tables are recorded by referring to their primary key attributes via foreign-key columns. This is enforceable with constraints, but only when the reference is never optional. Joins are computed at query time by matching primary- and foreign-keys of the many potentially indexed rows of the to-be-joined tables (Fig.40). These operations are compute and memory-intensive and have an exponential cost.

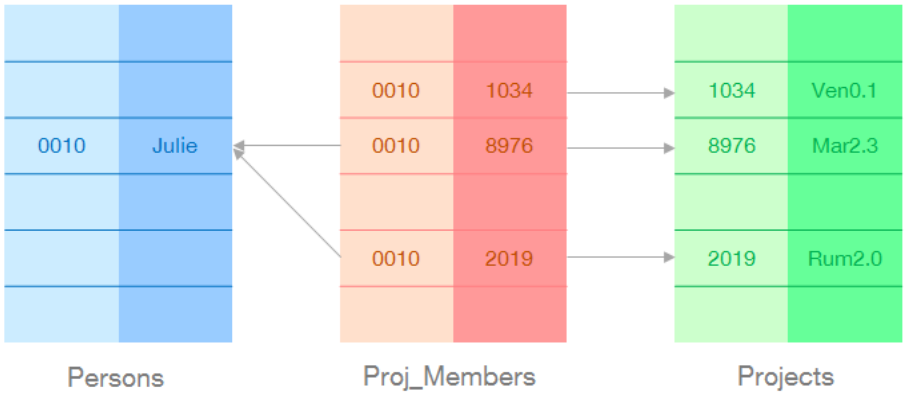

Figure 40 - Example of junction table to match people and project

When you use many-to-many relationships, you need to create a junction table (JOIN) that stores foreign keys of both participating tables which will of course increase join operation costs. Those costly join operations are usually solved by de-normalizing data to reduce the number of joins necessary.

These relational databases were originally created to diagram tabular structures and still yet are doing it very well. But contrary to what their name suggests, they are not effective for managing relationships between data. In particular when the structure of this data may vary, be adjusted…

This is probably the biggest weakness of relational databases: their lack of flexibility. In such a changing environment, constantly moving, such as software licensing models, evolving organizational IT processes, their scheme cannot support the dynamic real time, and uncertain nature of data, new technologies and platforms.

To overcome the lack of flexibility, you can divert the models. The challenge is to take into account all exceptions (non-modelized originally) and to strength embedding them to the original relational model. But this approach requires more code, energy, difficulties, and you have to resign to a simple and easy understandable model. Your data multiplies in complexity and diversity, your database is burdened with join tables which can reduce performance and paralyzed further developments.

However this relational model is still performant for many situations and it took a long time to emerge an alternative to this model.

2.2. From relation to Graph Databases

Graph data model are centered on relationships, unlike other database management systems, which require us to infer connections between entities using special properties such as foreign keys, or out-of-band processing like map-reduce. Just by connecting nodes and relationships, graph databases can create connected structures and so, sophisticated models that fit closer to our problem.

Each node (entity or attribute) in the graph database model directly and physically contains a list of relationship-records that represent its relationships to other nodes. These relationship records are organized by type and direction and may hold additional attributes. Whenever you run the equivalent of a JOIN operation, the database just uses this list and has direct access to the connected nodes, eliminating the need for expensive search / match computation.

This ability of pre-materializing relationships into database structures allows graph database to provide performances of several orders of magnitude, especially for join heavy queries, the minutes to milliseconds advantage that many users leverage.

The resulting data models in Fig. 41 are much simpler and at the same time more expressive than those produced using traditional relational or other NoSQL databases.

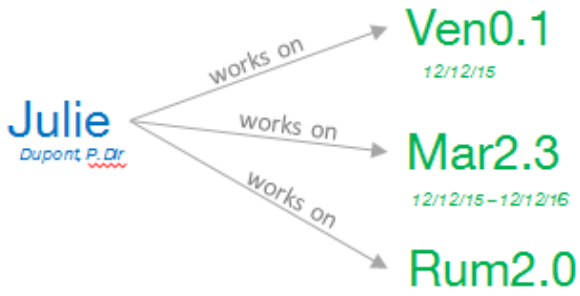

Figure 41 - Example of graph linking a person with projects

What is interesting for Software Asset Management is that graph databases support a very flexible and fine-grained data model which will allow us to model and manage this domain in a more intuitive and easier way. We can more or less keep the data as it is in the reality: unit, normalized, yet richly connected entities. This allows you to query and view your data from any imaginable point of interest, supporting many different use-cases.

The fine-grained model also means that there is no fixed boundary around aggregates, so the scope of update operations is provided by the application during the read or writes operation. The well-known and tested concept of transactions groups a set of updates of nodes and relationships into an atomic, consistent, isolated, and durable (ACID) operation. Graph databases like Neo4j fully support the transactional concepts including write-ahead logs and recovery after abnormal termination. So you never lose your data that has been committed to the database.

To manage software assets, we want a cohesive picture of our (big) data, including the connections between very different elements like contracts, installations and real usages. Contrary to relational databases, graph databases store data relationships as relationships. It means a lower disconnection between our evolving schema and our actual database.

By the facts, graph model is providing the flexibility which will allow adding new nodes and relationships without compromising any existing network. All original data (and its original relationships) remain intact.

Here is a quick recap of what is a graph database (property graph) (Fig. 42):

- A property graph contains nodes (data entities) and relationships (data connections).
- Nodes can contain properties.
- Nodes can be labeled with one or more labels.
- Relationships have both names and directions.
- Relationships always have a start node and an end node.
- Like nodes, relationships can also contain properties.

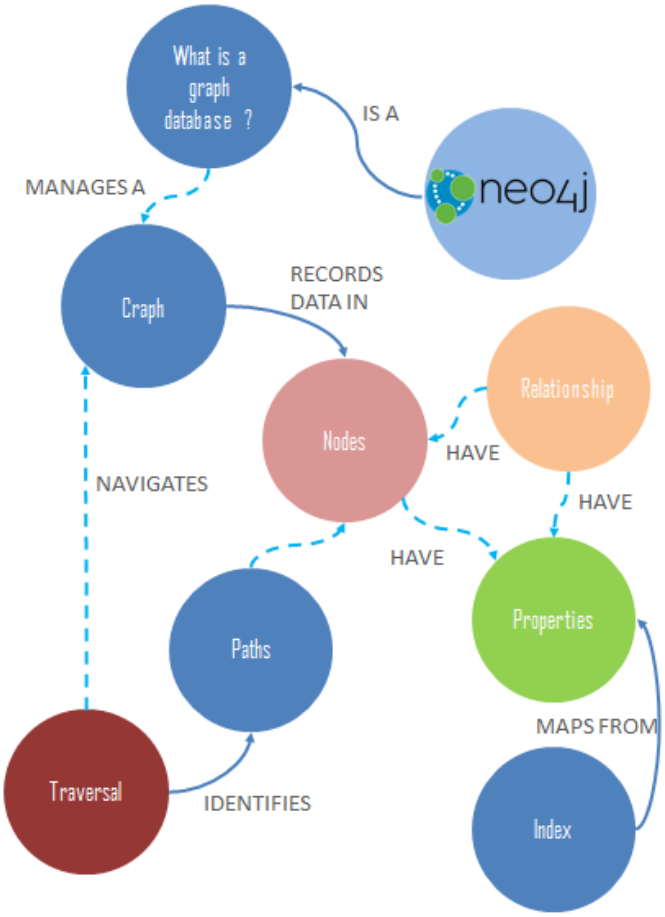

Figure 42 - Graph Database

Graph databases may be used as a tool for easy management of very intensively changing environment of data and data relations without losing focus on owner of asset and measurement of license demands.

Approach of storing asset data in graph databases is enabling fast and accurate allocation of software resources into project and initiatives. This way not only will help to keep usage of software according to license agreements, it will help to manage cost of software more precisely and forecast requirements. It will helps also to go beyond: to show in the future to software editors our true needs, asking for specific type of licensing, tailored exactly according to needs and model of usage of software in company.

2.3. SAM GRAPH PROPOSITION

Considering proposition of software and identification lifecycles and control loop proposed earlier, we propose a graph model in (Fig. 43) fulfilling loop requirements.

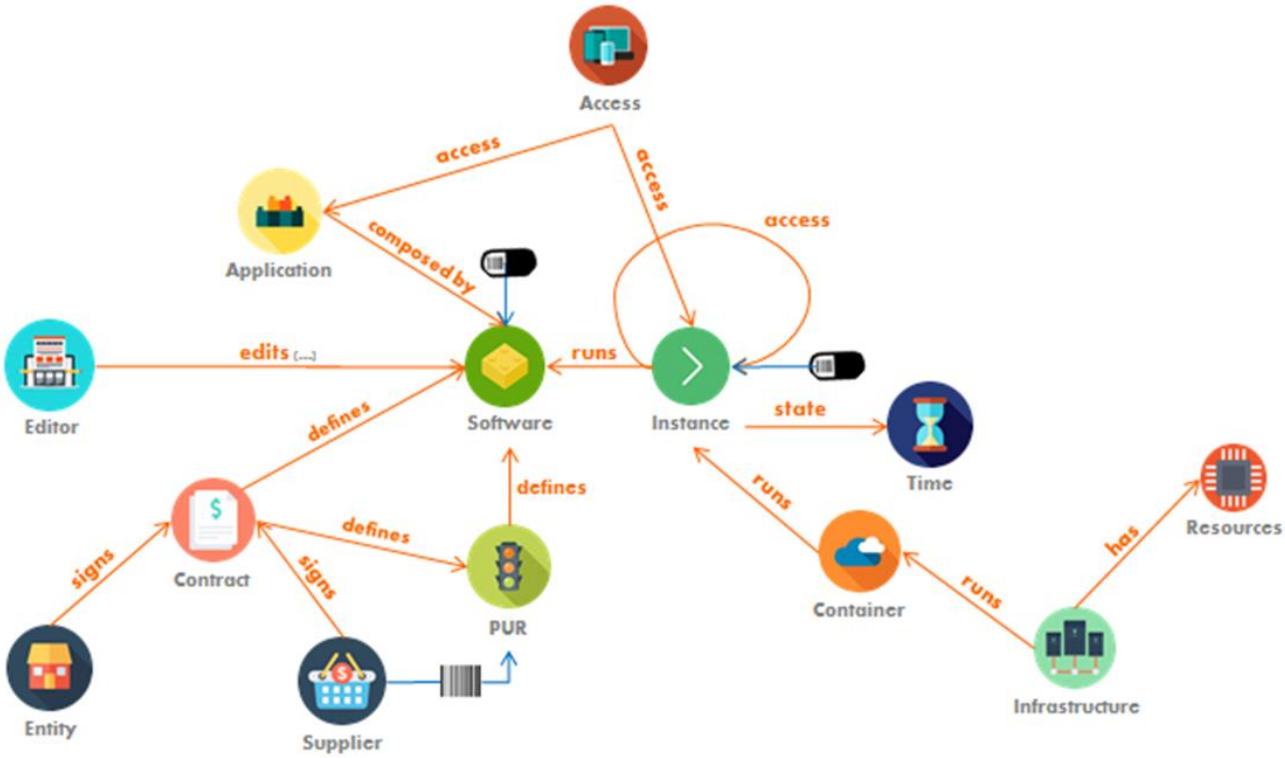

Figure 43 - SAM Graph Model

This graph models is motivated by real-life software lifecycle where component interconnectivity is a key feature. Here, information about data interconnectivity or topology is as important, as the data itself. In this case, the data and relations among the data are at the same level. Introducing graphs for this model has several advantages for this heterogeneous data:

It allows a more natural modeling of data. Graph structure is visible centered on the object 'Software' and allows a natural way of handling data. It has the advantage of being able to keep all the information about an object 'Software' in a single node and showing related information by arcs connected to it. Software is directly related to one or more contract(s), to its instances and to its usages which have to converge to check compliance and optimal deployment.

A contract is signed by an 'Entity' and a 'Supplier', it encompass several objects 'PUR' which represent all the specific licensing conditions agreed between the parties. Each specific PUR, corresponding to a unique label, has different properties depending it nature (i.e. metric, deployment restriction, access

conditions, etc.). The relation ‘defines’ between PUR(s) and software is characterized by the SKU.

The object ‘instance’ is bound to VLayers nodes which can be subdivided for each layer of virtualization up to the physical one. The relation ‘runs’ characterize effective consumed resources when each node contains specific workload features. Physical infrastructure is bound to resources (i.e. network board, cpu, hard drive …). Instance is bound to other instances to represent it lifecycle duration and encompass migrations, licensing persistence for backup architectures etc. The relation ‘runs’ between software and an instance is characterized by a SWIDTag containing a property ‘SKU’ which allow to bound an instance with associated PURs.

Access and uses can be monitored on a double level: software or application. The object ‘Application’ is composed by several software objects and depending the business purposes, this double monitoring is relevant.

Here, queries can refer directly to this graph structure. Explicit graphs and graph operations allow the SAM to express a query at a high level of abstraction. (i.e. “which entity(ies) own this specific software ?”,”which PURs characterize my instance”?) To some extent, this is the opposite of graph manipulation in deductive databases. It is not important to require full knowledge of the structure to express meaningful queries.

We propose to use this graph model to develop further SAM sensors and effectors in section below, based on two specific use-cases.

To design SAM controls we need to make assumptions on the targeted cloud environments, especially in terms of the IaaS/PaaS layer that will be used to deploy services. In a first design, we consider clouds managed through the well-known and commonly used Cloud Foundry PaaS [32]. We consider that it will be possible to apply our model to a variety of PaaS as long as they allow instantiation/usage’s capture. In a second layer, we consider an Infrastructure as a Service (IaaS) approach based on BluePlanet (Network Orchestrator designed by Ciena) plugged on an OpenStack plateform. Both of these approaches are in use in Orange, but also quite representative of global trends in industry and telcos.

The main difference between the first approach – Pure PaaS – and the second that we will present here is that a container is a way of packaging an application and all its dependencies into a single entity that can be run on a Linux or other server. It is similar to a virtual machine (VM), but lighter-weight than a VM because a container doesn’t include an operating system. Multiple containers (each running different application) may be run on any given VM. A hypervisor is software that

creates, runs, and monitors VMs. That being said, from SAM point of view, the difference is slender, in both case, access to the valuable configuration data will be perform through the monitoring interface.

We propose, in next chapter, to describe two use-cases (1) PaaS Instantiation and usage capture, (2) IaaS infrastructure for network virtualization management and to propose as syntheses SAM controls for both use-cases.

Chapter 5

V. MODEL ASSESSMENTS

V. Model Assessments 130

1. Platform Use Case I: PaaS - Cloud Foundry 132

1.1. Cloud- PaaS instantiation and usage capture 132

1.2. How does Cloud Foundry work? 133

1.3. Instances 134

1.4. Usage collect 135

1.5. Modeling 137

a. Graph model Construction 139

i. Purchasing 139

ii. Provisioning 144

III. Instantiation 145

iv. Usages 148

b. Basic control of inventory's consistency 150

c. Cost-Saving Identification 153

2. Platform Use Case II: Network Function Virtualization 157

2.1. Orchestration & Hypervisor in Operator's network 157

2.2. Context concerns 162

2.3. Usage collection 165

2.4. Modeling 165

a. Graph model construction 168

i. Purchasing 168

ii. Provisioning 173

III. Instantiation 173
iv. Options 176
b. Cost-saving Identification 177

We propose a qualitative evaluation of our model to prove it usability in modeling cloud platforms and resources, software deployments; and guarantee legal compliance and cost optimization no matter the licensing model level of complexity.

We underlined the necessity to build a multiplatform inventory solution to track all assets, their hardware configuration, software deployments and their virtual resources and usages regardless of platform. We need to identify each virtual appliance that run for couples of minutes of days and that cannot be find by any scheduled inventory scans or agent deployments but directly from getting this information from the build process.

1. PLATFORM USE CASE I: PAAS - CLOUD FOUNDRY

Many enterprises adopt a PaaS platform such as Cloud Foundry (CF) [32] to enable easier scaling and management of applications. Cloud Foundry is an open source project originally started by VMWare and now owned by Pivotal which is a joint venture of VMWare, GE and EMC. PaaS platform such as CF enables developers to focus on development and provides entire platform at click of a button. Developers can simply deploy their binary archives and CF takes care of provisioning everything required for application to run. CF also provides additional components such as database, caches as a service which makes it a true platform.

1.1. CLOUD- PAAS INSTANTIATION AND USAGE CAPTURE

From our observations in literature and from Orange experience, we already reach this statement how complex SAM in dynamic cloud environments is. In fact, just answering the following questions is difficult even though it is crucial for any organization who wants to rationalize its IT expenditures:

- What is the total number of virtual/physical devices on the network?
- How many devices and apps are deployed in any datacenter whatever is underlying technology and environment?
- What software is being used and what is lying redundant?
- Which users are using which devices to access company applications?

Our experience shows that for sure, companies have some visibility of what is on the network, but mainly from a number of different and disparate sources. We underline that the real need is to have a single source giving a complete view across

the entire IT environment. The unknown represents a massive risk and cannot be proactively managed. Fact is that Gartner[16] predicts that 'by 2020 large enterprises with a strong digital business focus or aspiration will see business unit IT increase to 50% of enterprise IT spending'. And, according to Forrester's recent publication[17] of its Midyear Global Tech Market Outlook, the trend for Cloud Adoption is accelerating (to 5.6%) as well as software being the second-largest category of tech spending (after telecom services).

Orange, like other telco and famous digital companies are intensifying move to the Cloud and hardly evaluate the potential financial risks like software license cost's explosion. They have to handle these costs, and discover and inventory their entire estate in order to be able to make decisions on future plans and get optimizations. Even if we want to focus on license expenditures, we need to consider for a moment the combined costs of hardware and software assets to understand how cloud technologies, which encourage insatiable consumption, can create unused cloud licenses and virtual hardware which are left running and leads to another cause of overspend.

We understand the necessity to build a multiplatform inventory solution to track all assets, their hardware configuration, software deployments and their virtual resources and usages regardless of platform. We need to identify each virtual appliance that run for couples of minutes, of days and that cannot be find by any scheduled inventory scans or agent deployments but directly from getting this information from the build process.

1.2. How does Cloud Foundry work?

Clouds balance their processing loads over multiple machines, optimizing for efficiency and resilience against point failure. A Cloud Foundry installation accomplishes this at three levels:

1. BOSH creates and deploys virtual machines (VMs) on top of a physical computing infrastructure, and deploys and runs Cloud Foundry on top of this cloud. To configure the deployment, BOSH follows a manifest document.

---

[16] Gartner Inc. Metrics and Planning Assumptions Required to Drive Business Unit IT Strategies. 21 April 2016. Analysts: Kurt Potter | Stewart Buchanan

[17] Forrester Research. The Midyear Global Tech Market Outlook For 2016 To 2017. Slowing Economies And Cloud Constrict Tech Market Growth. September 16, 2016. Analysts Andrew Bartels with Matthew Guarini, Rachael Klehm

2. The CF Cloud Controller runs the apps and other processes on the cloud's VMs, balancing demand and managing app lifecycles.

3. The router routes incoming traffic from the world to the VMs that are running the apps that the traffic demands, usually working with a customer-provided load balancer.

Cloud Foundry designates two types of VMs: the component VMs that constitute the platform's infrastructure and the host container that host apps for the outside world. Within CF, the Diego system distributes the hosted app load over the entire host containers, and keeps it running and balanced through demand surges, outages, or other changes. Diego accomplishes this through an auction algorithm.

To meet demand, multiple host containers run duplicate instances of the same app. This means that apps must be portable. Cloud Foundry distributes app source code to containers with everything the containers need to compile and run the apps locally. This includes the OS stack that the app runs on, and a buildpack containing all languages, libraries, and services that the app uses. Before sending an app to a container, the Cloud Controller stages it for delivery by combining stack, buildpack, and source code into a droplet that the VM can unpack, compile, and run. For simple, standalone apps with no dynamic pointers, the droplet can contain a pre-compiled executable instead of source code, language, and libraries.

1.3. INSTANCES

Deploying an application through the Cloud Foundry (CF) PaaS layer is done by running a push command from a Command Line Interface (CLI), either as part of the CF build packs or through a service broker:

- Build pack. User pushes app bits (i.e. artefact: .jar, .war, tgz, etc.) from desktop/CLI selecting one of the supported stack (i.e., Ubuntu)
- Service broker pushes a docker image reference (public or private registry), or a container specification reference

In both cases, a droplet is produced, taking into account dependencies configuration; As a result, app instances are started and run the image within quotas (Random Access Memory (RAM), Computer Process Unit (CPU), etc.).

Among others, between push and application's availability, CF uploads and stores the application files, and examines and stores the application's metadata (for SAM purposes the software identifier enriched by all relevant contractual information during delivery step).

As the cloud operates, the Cloud Controller VM, router VM, and all containers running apps continuously generate logs and metrics. The Loggregator system aggregates this information in a structured, usable form, the Firehose. You can use all of the output of the Firehose, or direct the output to specific uses, such as monitoring system internals or analyzing user behavior, by applying nozzles.

Before one can retrieve any application or service information, one must retrieve the Cloud Controller (using the Service Broker Application Programming Interface (API)). The brain of this controller knows services and applications as well as their instances and settings. The Cloud Controller exposes a Rest (Representational State Transfer) API for all this information through which the SAM processes can get the necessary knowledge to perform their tasks.

1.4. USAGE COLLECT

To organize user access to the cloud and to control resource use, a cloud operator defines Orgs and Spaces within an installation and assigns Roles such as admin, developer, or auditor to each user. The User Authentication and Authorization (UAA) server supports access control as an OAuth2 service, and can store user information internally or connect to external user stores through LDAP or SAML.To implement SAM check-points over the USAGE step, we need to get the knowledge of which applications are used. We decided to achieve this first through the application rights verification. In more details (Fig. 44), we summarize the steps performed when a user wants to use an application in our context:

1. The user wants to access the cloud application via the user portal
2. The user is identified and authenticated via a User Identification and Information System Access libraries
3. The system checks permission of the authenticated user to access the applications via the Application rights library and if yes, return a certificate. This step allows collecting usage information, especially the moment when a certificate for using the application is issued or withdrawn. The lifecycle of this certificate allows determining the time of using the application and all its software components

4. Embedding cookies and certificate, the user can start to consume application

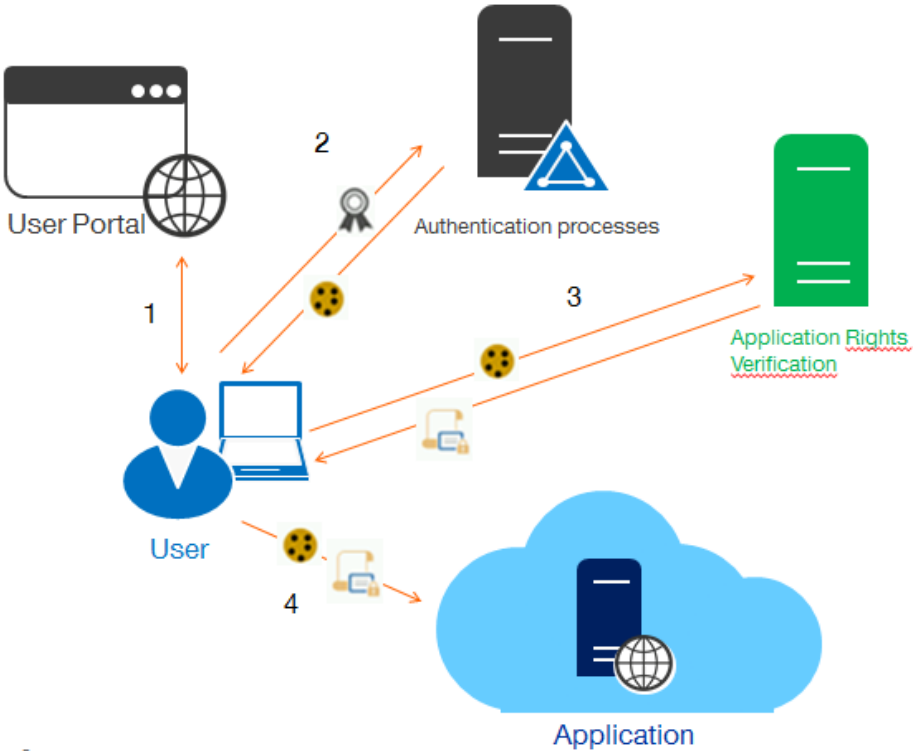

Figure 44 - Use case of Cloud App Access

An application may embed several software services, so it is necessary to cross the information on usage with internal software cartography to be able to determine and affect usage directly to software. Typical apps depend from services such as databases or third-party APIs. To incorporate these into an app, a developer writes a Service Broker, an API that publishes to the Cloud Controller the ability to list service offerings, provision the service, and enable apps to make calls out to it.

Application's usages cannot be summarized only by a number of access or minutes spent. We consider that it also covers consumed resources (i.e., CPU, RAM, bandwidth, event p/s, flow p/s, etc.).

Open-source tool Abacus [33] provides usage metering and aggregation for Cloud Foundry services (Fig. 45). This is implemented as a set of REST micro-services that collect usage data, apply metering formulas, and aggregate usage at several levels within a Cloud Foundry organization. Runtime provider (CF

Bridge) submits application usage events (other runtime providers submit other runtime usage events); external services providers submit service usage events that are received and stored by Abacus, metered, accumulated, aggregated to provide usage reports and summaries.

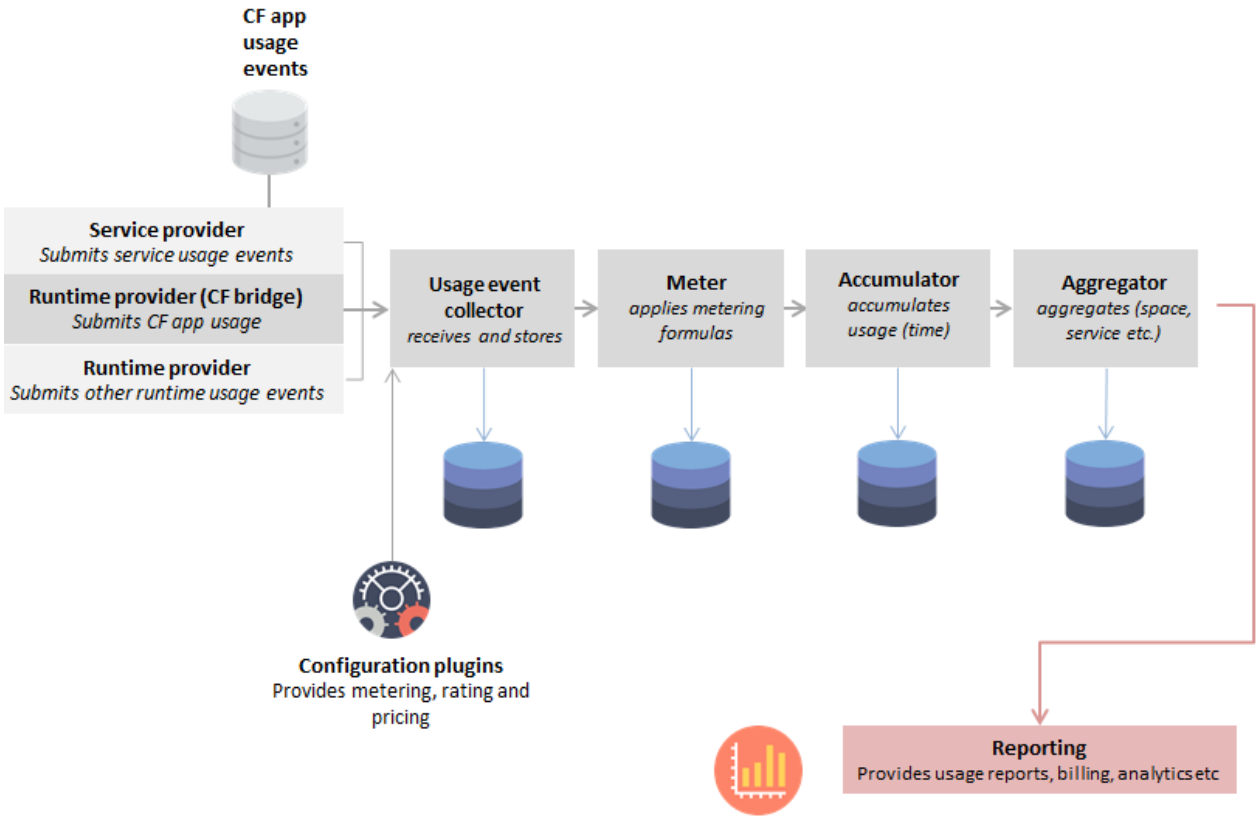

Figure 45 - Usage metering and aggregation for Cloud Foundry

1.5. MODELING

Our objective is to validate the fact that our graph model can be managed through a capture of PaaS usages (Cloud Foundry/Abacus). It will validate our assumption that the model can be used to easily model complex platforms and software. To achieve these experiences, we first considered well-known software: an Oracle database (Oracle DB).

We choose the Oracle Database Enterprise Edition (Oracle DB EE) example for several reasons:

- It is a vivid example for the SAM community; one of the most often mentioned for the complexity of its license management.
- Oracle DB licenses can be defined by several types of metrics, oriented on material (i.e., CPU) or user (i.e., Named User Plus).
- It will allow us to increase complexity of our use cases such as: integrating controls between product's link (options – standard product) and constraints of uses.

To evaluate relevance of our model, we propose to define in Fig. 46 below a Cloud architecture model.

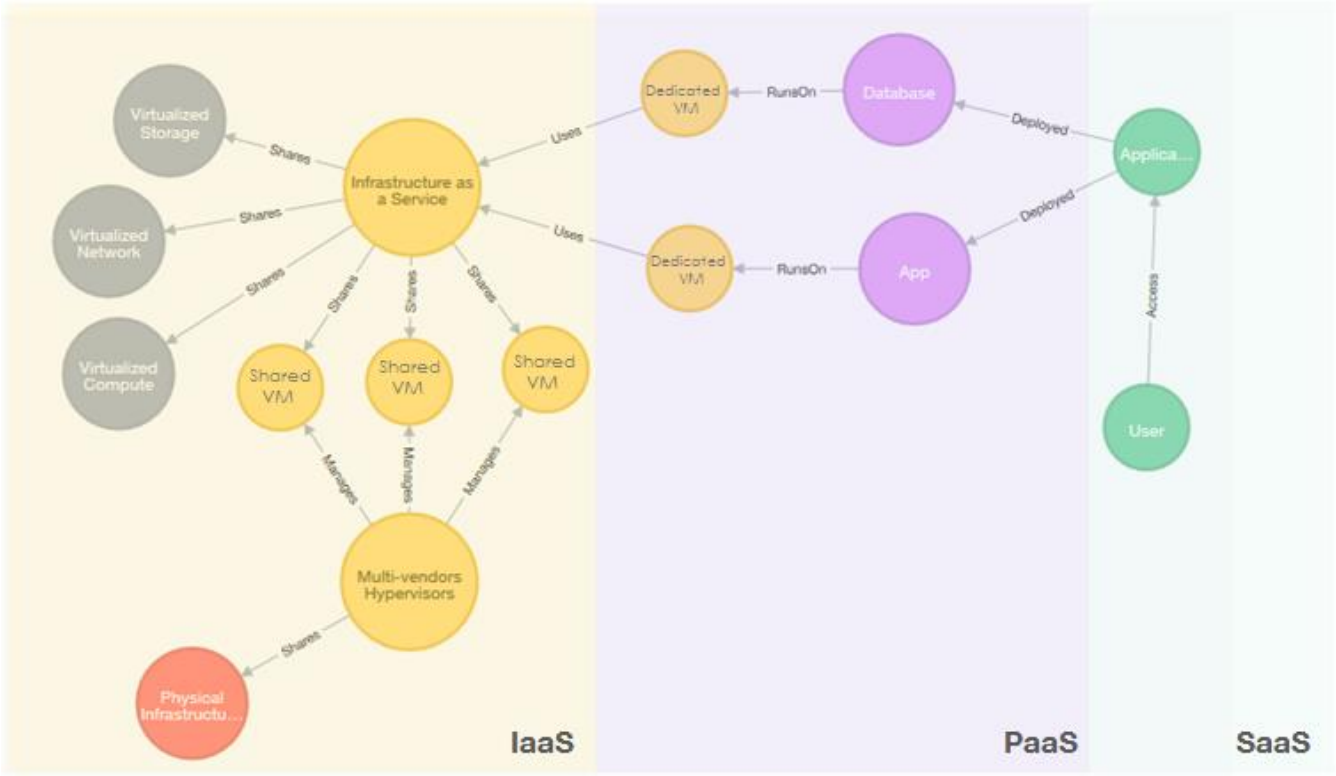

Figure 46 - Cloud Architecture Model

The diagram illustrates how the PaaS will leverage enabling services from IaaS components and supporting services to provide PaaS service offerings, such as application hosting and database hosting. The IaaS model is depicted by a node labeled 'Infrastructure as a Service' that contains compute virtualization, network virtualization and storage virtualization. (Not detailed in the diagram, the

'Virtualized Compute' node contains smaller nodes labeled CPU and Memory as an illustration of the technologies that make up compute virtualization. The 'Virtualized Network' node contains smaller nodes labeled Network Interface Card, TCP/IP Ethernet, Fibre Channel on Ethernet and Load Balancer to illustrate technologies that make up network virtualization. The 'Virtualized Storage' node contains smaller nodes labeled Thin Provisioning, Block, Object, Solid State Drive and Serial Advanced Technology Advancement to illustrate technologies that make up storage virtualization). The hypervisor label is associated with this pattern in its commitment to applying reservations for different cloud consumers in order to ensure that they are allocated the guaranteed amounts of IT resources. In support of this, it is responsible for locking, and pre-allocating the virtual servers' reserved computing capacity based on their configurations. The PaaS model is depicted in violet box that contains a PaaS stack, Application hosting Stack. The Application Hosting stack is comprised of layered nodes labeled 'App server', 'database server'; we could add 'Web server' or 'operating system'.

a. Graph model Construction

In this section, we will follow the Software lifecycle proposed in previous chapter and refer to the Fig.2 about SAM maturity scale: Visibility, Identification, Risk Management and Optimization.

i. Purchasing

For the purpose of our example, we will skip the first phase of need/choice/approval, and directly start with purchasing processes.

Fig.47 might be an extract from "License Store's" catalogue which proposes the product we identified as needed and are planning to buy.

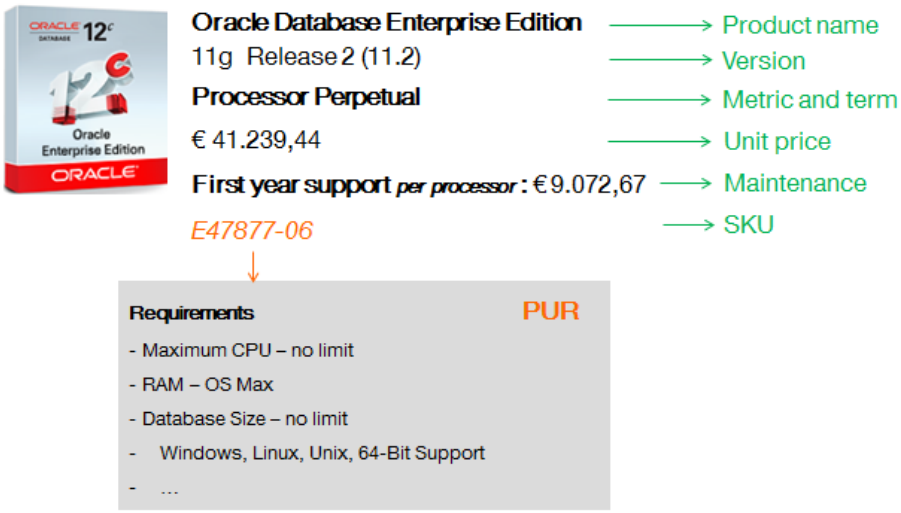

Figure 47 - Product Catalog

Few elements (in green above) are necessary to identify precisely this offer and determine the level of grants (PUR) given by this type of licensing. These elements have to be collected in the purchase order and reconciliated with data from the delivery order.

In the graph, on Fig.48, the first step is to create our product, with a label 'Software' and several attributes found in the purchase order. In the same way, we create a label 'Supplier' and 'Editor' to identify a node 'License Store' and 'Oracle':

Figure 48 - Neo4J interface - Graph Step 1

```
CREATE (S:SOFTWARE {NAME:"ORACLE DATABASE ENTERPRISE EDITION", SKU:"E47877-06",VERSION:"11G RELEASE 2", CATEGORY:"DATABASE"})

CREATE (R:SUPPLIER {NAME:"LICENSE STORE"})

CREATE (E:EDITOR {NAME:"ORACLE"})
```

Then, Fig. 49, we create several nodes with label 'PUR', which represents scope of usage, metrics, environments … The idea is to create nodes, independent from products (not node properties) to allow further comparison between product, version; identify similar metrics; verify entitlement compliance.

```
CREATE (P:PUR {TYPE:"METRIC",METRIC:"PROCESSOR"})

CREATE (P1:PUR {TYPE:"TERM",TERM:"PERPETUAL"})

CREATE (P2:PUR {TYPE:"REQUIREMENT",MAXIMUMCPU:0})

CREATE (P3:PUR {TYPE:"OPERATING SYSTEM", WINDOWS:1, LINUX:0, UNIX:0})
```

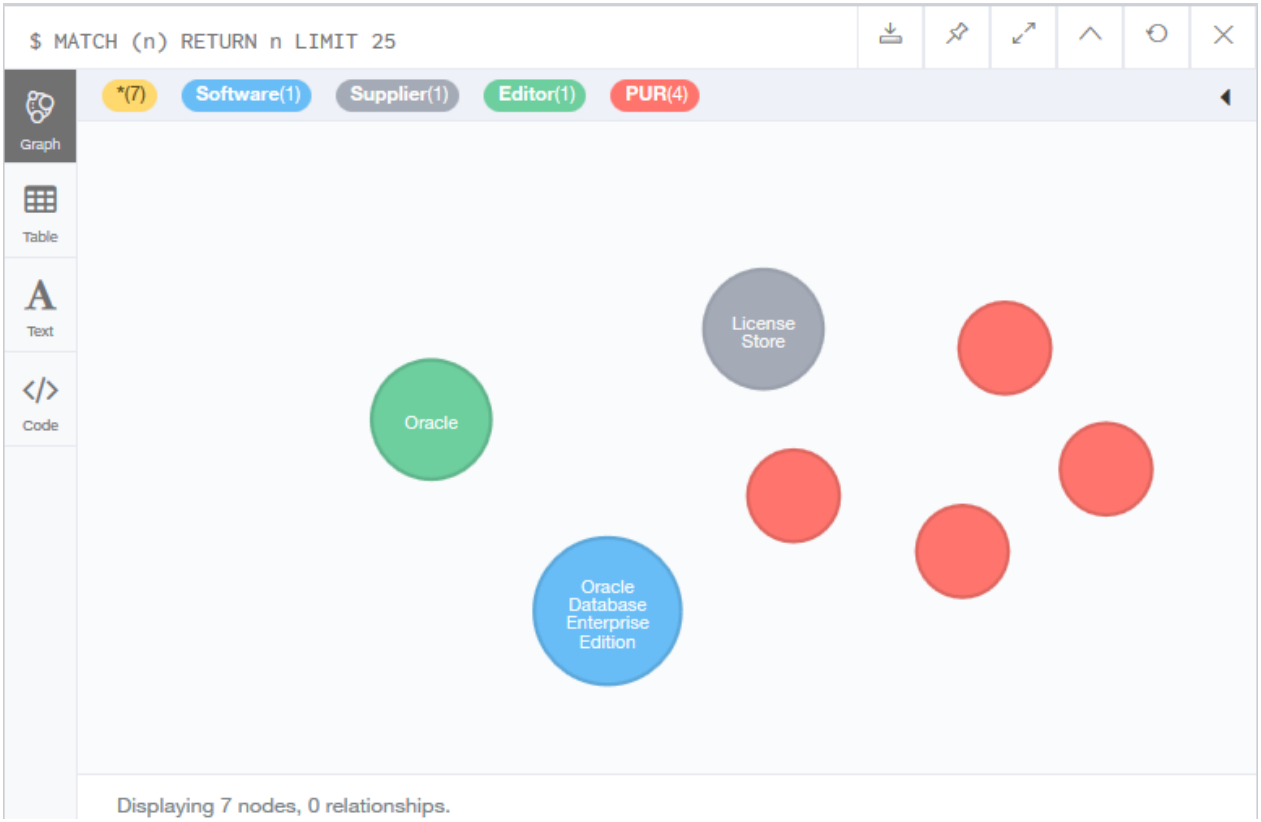

Figure 49 - Neo4J interface - Graph Step 2

Then: to create relations between nodes:

- Between an editor and product (EDITS): 'Oracle' edits 'Oracle Database'
- Between a product and PUR (DEFINES): 'Oracle DB' is licensed under processor metric/ or can run on windows/Unix/Linux …
- Between a supplier and a product (DISTRIBUTES): 'License Store' distributes 'Oracle DB'. This relation is important because contains all information about the contract: financials, number, maintenance, etc. This link may be multiple (unique relations), as many as the number of contract.

This process and collect are essential to fulfill Identification requirements: PUR are translated in the SKU, this SKU enriches the SWIDTag delivered during provisioning processes; it guarantees the link between a contract and Software/ Software and Instance, on Fig.50.

```
MATCH (s:SOFTWARE {NAME:"ORACLE DATABASE ENTERPRISE EDITION"})
```

```
MATCH (u:Supplier {name:"License Store"})
MATCH (e:Editor {name:"Oracle"}) MATCH (p:PUR)
MERGE (c:Contract {name:"CSI001",date:01-06-2017,contact:"Felix"})
MERGE (sp:Entity {name:"Service Provider"})
MERGE (p)-[d:DEFINES]->(s)
MERGE (e)-[e1:EDITS]->(s)
MERGE (u)-[d1:DISTRIBUTES]->(s)
MERGE (u)-[s1:SIGNS]->(c)  MERGE (sp)-[s2:SIGNS]->(c)
MERGE (c)-[d2:DEFINES {Quantity:10, unitPrice:150, currancy:"£"}]->(s)
```

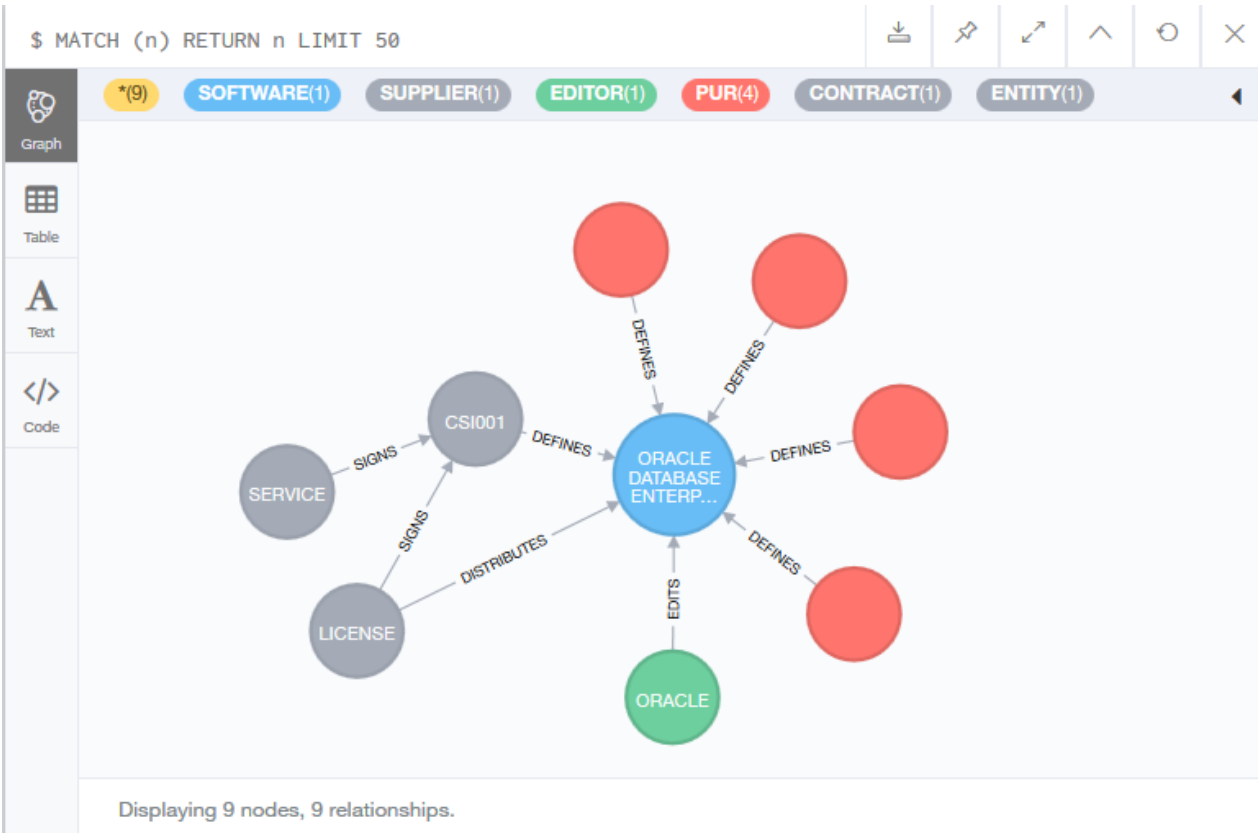

Figure 50 - Neo4J interface - Graph Step 3

ii. Provisioning

After Global sourcing processes, our Oracle Database is right now under exploitation teams' responsibility. The software can be packaged/enriched (i.e., tag) according to company's rules or considered like included in an Application before being instantiated.

In our case, we create a label 'Application' and a node 'HumanRessources' which will include our Database. An application program is a computer program designed to perform a group of coordinated functions, tasks, or activities for the benefit of the user, composed by one or several software. Each application has its own SKU characterizing its composition and eventual licensing models (in case where the service provider will propose it as a commercial offer).

The relations 'CONTAINS' is enriched by properties like a project's id or application's project manager (Fig. 51)

```
MATCH (S:SOFTWARE)

CREATE (A:APPLICATION {NAME :'HUMANRESSOURCES', SKU:"AA90875"})

CREATE (A)-[C:CONTAINS  {ID_PROJECT : '1234R'}]->(S)
```

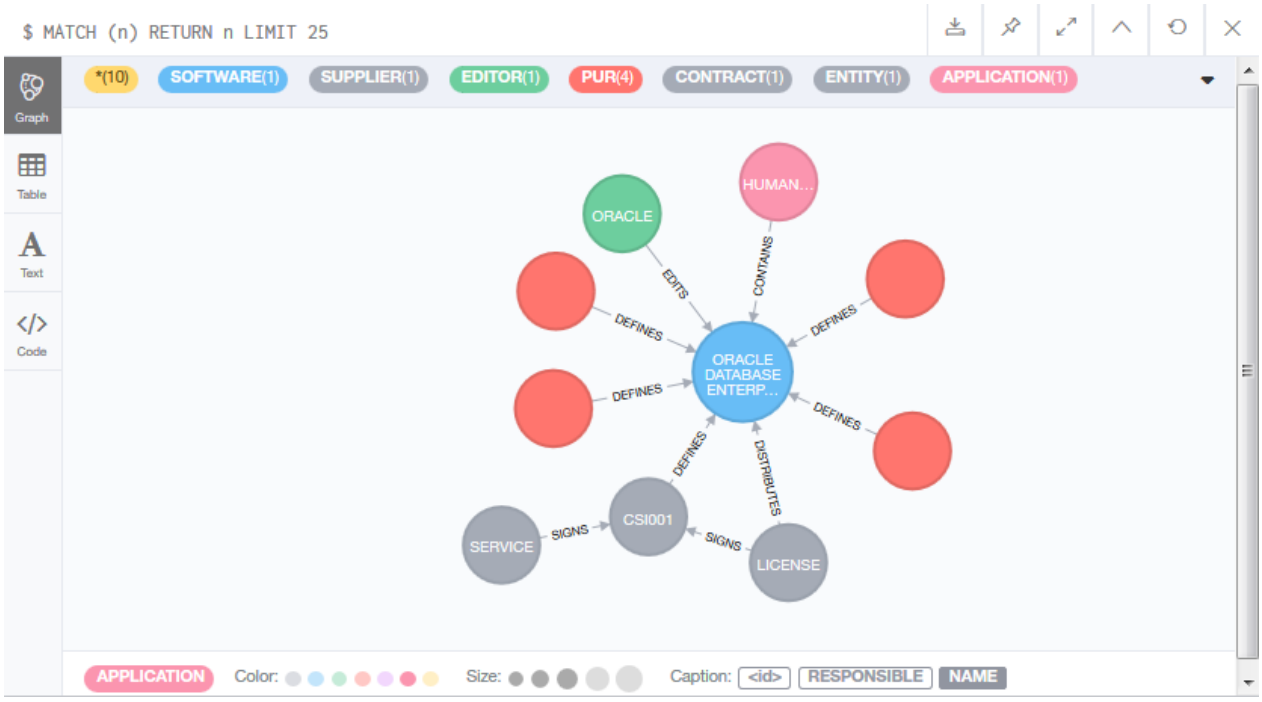

Figure 51 - Neo4J interface - Graph Step 4

iii. Instantiation

To fulfill the step 1 (*visibility*) of the maturity scale, we need to have an exhaustive view of infrastructure, resources and instantiation. The PaaS handles infrastructure workload (Virtual Machine (VM), networking, storage); instantiation's inventories, subscription to shared services, application deployment, installation, configuration, application monitoring, application log collection and interaction with app-ops (inventory/CMDB, monitoring/alerting).

Crucial point is now to create a link between the instance and the product which we bought. The instance knows and updates all identification elements of it components. This allows creating the link between the product in catalogue and the installed product, Fig. 52.

```
MATCH (S:SOFTWARE{SKU:"E47877-06"})

MERGE (I:INSTANCE {NAME:"INSTANCE DB",SKU:"E47877-06",IMAGE:"DBENTREPRISEEDITION"})

CREATE (I)-[I1: INSTANTIATES] ->(S)
```

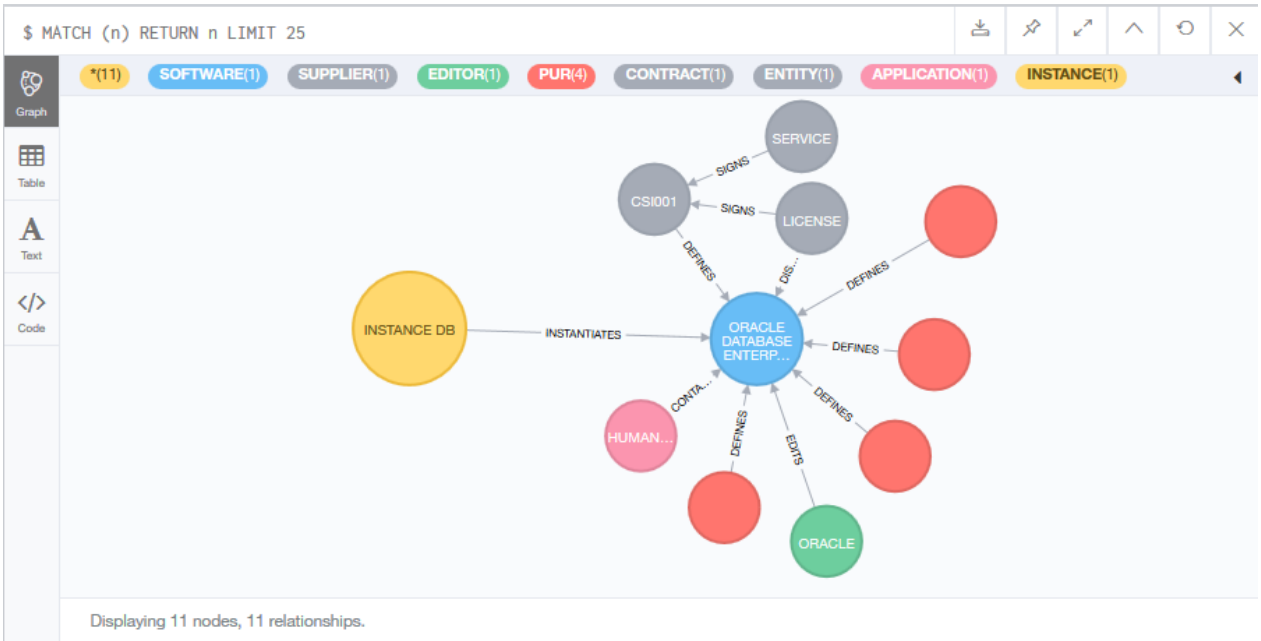

Figure 52 - Neo4J interface - Graph Step 5

The characteristics of software, its instances and their configuration have to be traceable regardless of the deployment conditions. This requirement of

traceability encompasses precise identification of Software and resources to allow maintenance of deployment inventories. IaaS (Open Stack) offers bare HW resources access. The bridge developed through abacus metering solution allows binding PaaS deployment to physical architecture using IaaS resources collection.

In our example, the application, which contains our Database has been deployed on the cloud via a “push” command and ran as an instance. We stress that this instance’s image contains metadata used for identification enclosed during the provisioning (Fig.53). To make it simpler for the purpose of this example, we will transform it into a property ‘SKU’ of the Instance.

```
MATCH (I: INSTANCE)
MERGE (VM:CONTAINER {NAME:'VM1',CPU:4, RAM:8})
MERGE (T:TENANT {NAME:'TENANT1',NBINSTANCE :20, RAM :100})
MERGE (V:VLAYER {NAME:'OPENSTACK1', REGION:'FRANCE', VERSION:'ZOE'})
MERGE (M:MACHINE {NAME:'BAREMETAL'})
MERGE (R:RESOURCE {TYPE:'CPU', RAM:'X86'})
CREATE (VM)-[R0:RUNS]->(I)
CREATE (T)-[R1:RUNS]->(VM)
CREATE (V)-[R2:RUNS]->(T)
CREATE (M)-[R3:RUNS]->(V)
CREATE (M)-[R4:HAS {NUMBER:20}]->(R)
```

Considering the Fig. 39, we are collecting and integrating data corresponding to allocated resources (the green branch), technologies and binding it to eventual restriction.

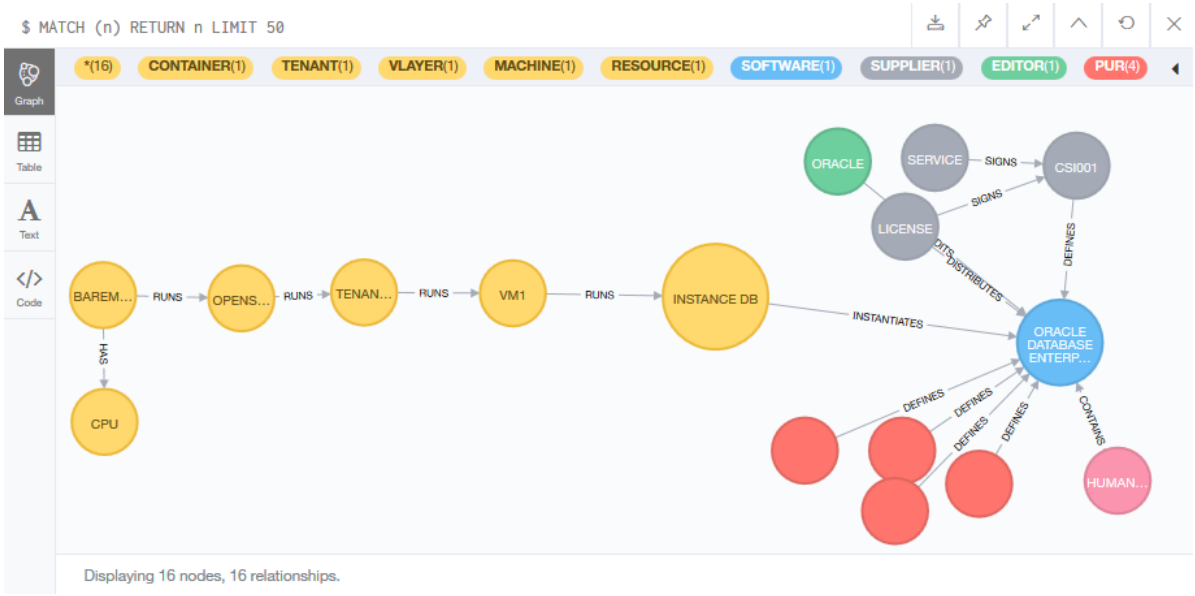

Figure 53 - Neo4J interface - Graph Step 6

In case of migration, Fig. 54, we need to keep and maintain a history for auditing purposes. We propose to create a relation between instances specifying that relatedness.

```
MERGE (I1: INSTANCE {NAME:'INSTANCE DB', STATUS:'MIGRATION'})

MERGE (I)-[R5:MIGRATEDTO {TIMESTAMP:15908664663}]->(I1)
```

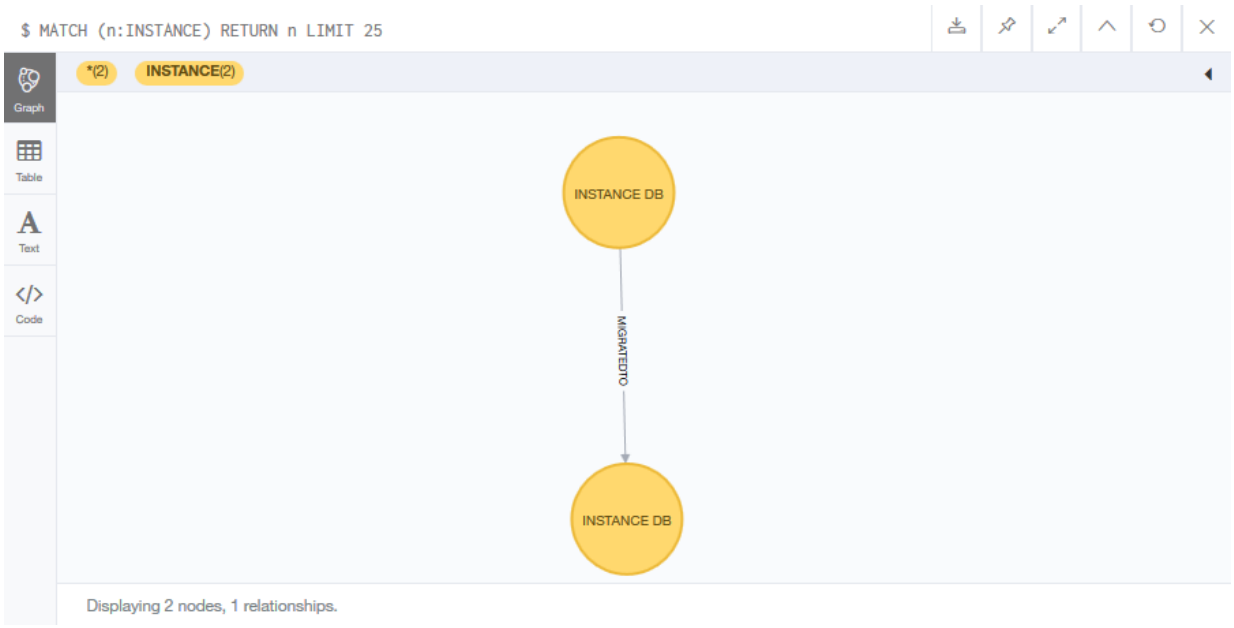

Figure 54 - Neo4J interface - Graph Step 7

iv. Usages

The Oracle DB is expected to be accessed by both humans and Software (automated applications encompassing the optimization phase of the SAM model as described previously in the paper). Different queries can be performed on the different links of the database.

Considering the Fig. 39, we are collecting and integrating data corresponding to consumption (blue branch) and supervision (red branch); In terms of access, object, service, traffic, time, and resources, network size, amount of managed objects, etc. Each event, captured is enriching a relation property. We make a difference between theoretical capacities and observed ones. For example, the relation HAS between a machine and its CPU is characterized by two properties:

- Formal link between the machine and CPU : the machine HAS 20 CPU
- The machine HAS real observed CPU consumption data (coming from the current CPU load observation)

We propose to modelize it in Fig. 55, using a mesure of uses based on access (consumption) and Authaccess (supervision/consumption).

```
MATCH (I:INSTANCE {NAME:'INSTANCE DB'})

MERGE (U:USER {NAME:'JEAN', ID:'1906197913022014'})
```

```
CREATE (U)-[A:ACCESS {CHARACTERISTIC:'RESOURCES'}]->(I)

---

MATCH (AA:APPLICATION)

MERGE (U:USER {NAME:'CATHERINE', ID:'12071962'})

CREATE (U)-[A:AUTHACCESS {CHARACTERISTIC:'RESOURCES'}]->(I)

CREATE (U)-[AAA:AUTHACCESS {CHARACTERISTIC:'RESOURCES'}]->(AA)
```

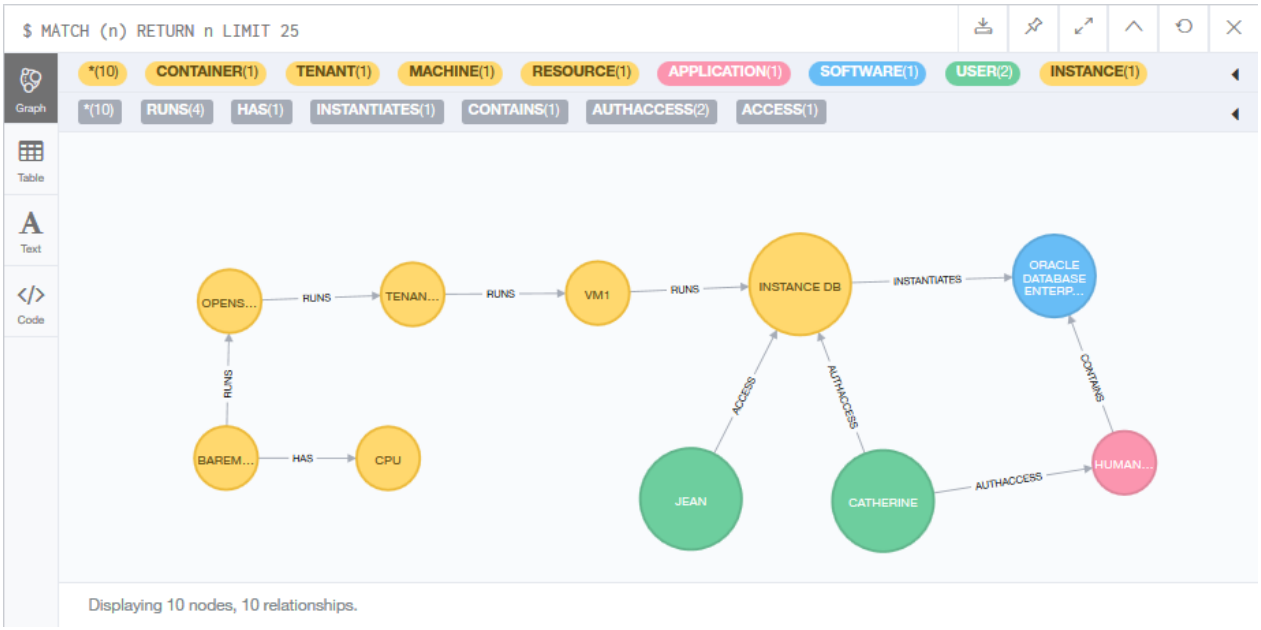

Figure 55 - Neo4J interface - Graph Step 8

"Show me all 'ACCESS' relation(s) to 'HumanRessources" will provide all access/authaccess related to Software. As we can identify the Product Usage Rights (via the SWIDTag/SKU) by a direct link between Software/PUR and Software/Access, we can fulfill second part of the step 3 (here: over-deployment risk).

b. Basic control of inventory's consistency

Obviously, a lot of queries would be necessary to implement true SAM analysis. For the purpose of our example, let's study quickly three of the most basic, but also the most important.

- What I bought?

This query (Fig. 56) returns a table: the number of bought licenses order by software and metric with a list of contract per software

```
MATCH(E:ENTITY)-[G:SIGNS]-(C:CONTRACT)-[H1:DEFINES]->(S:SOFTWARE)

MATCH (P:PUR{TYPE:'METRIC'})-[H:DEFINES]->(S)

RETURN E.NAME AS ENTITY, S.NAME AS SOFTWARE, S.SKU AS SKU, P.METRIC AS METRIC, H1.QUANTITY AS QUANTITY
```

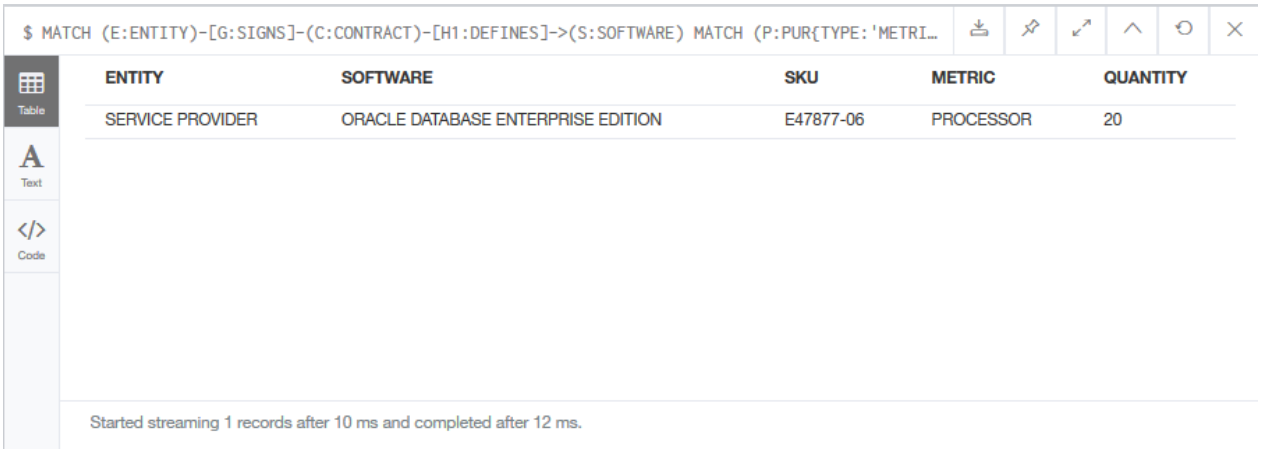

Figure 56 - Neo4J interface - Query Bought

- What I Instantiated?

This query (Fig. 57) returns a table: the number of instance per software ordered by metric with collection of application containing this software.

```
MATCH (V:CONTAINER)-[T:RUNS]->(I:INSTANCE)-[R:INSTANTIATES]->(S:SOFTWARE)<-[]-(A:APPLICATION)
```

```
RETURN S.CATEGORY AS CATEGORY, S.NAME AS SOFTWARE, A.NAME, S.SKU
AS SKU, COUNT(T) AS INSTANCENUMBER
```

Figure 57 - Neo4J interface - Query Instance

- Am I Compliant?

In our simple example, this last query (Fig.58) consists in a verification of the 'Processor' metric (typical for Oracle). Basically, we have to multiply number of core per processor of the physical machine hosting the DB by the number of processor and by a coefficient given by Oracle for each processor. It returns a table of licenses ordered by software, number of bought/instantiated (according to Oracle licensing rules).

Figure 58 - Neo4J interface - Query Compliance

Some libraries like, Popoto.js (Fig.59) allow user to navigate in the graph to do the same in graphic mode. Popoto is an open-source library.

Figure 59 - Popoto for graphic Neo4J interface

Optimization consists first in automating the rise of alerts. When counterfeiting situation is detected (piracy, but mainly editor's metric misunderstanding) or when use reaches or exceeds a fixed threshold or the level of inventories. Then, purchasing/activating new licenses could be automated to adjust the license stock, in real time.

When the visibility and identification steps are mastered, optimization might consist in operating simulations: usage/instantiation captures, may reveals some possibility to renegotiate a contract in a more favorable (financial) way: i.e., to change the Oracle DB negotiated metric (currently Processor) into another metric (i.e., Access), more appropriated to observe uses. Or to project a future software/license uses based on current observed situation.

c. Cost-Saving Identification

Based on the previous observations and experimentations, we built a prototype (cSAM) in order to analyze real and dynamic uses of software resources in cloud environments. First to ensure compliance and to determine real costs for users, then to optimize the deployment of licenses based on predefined and adjustable scenario. It is based on tag process recognition and implementation for software identification purpose and modeling based on graph.

Comparing to SAM market tools, the prototype developed in NodeJS shows it value-added in integration of cloud dynamicity issues; it is flexible and multi-domains; it is design (Fig. 60) to integrate quickly and easily new/complex metrics linked with new business models and support innovative simulation functions to allow better uses and deployment control. The simulation functions encompass possibility to simulate a change in metric and evaluate the best in terms of licensing costs depending the inventoried deployments and uses. Moreover it add the possibility to evaluate the impact of changing allocated resources in terms of licensing costs. cSAM relies on several asynchronous sources of provisioning related with software lifecycle: First one is from procurement information system (the weakest part of our experiment, due to the difficulty to access some confidential data from contracts and the arcane legacy of sourcing information system we were not yet able to automatize relaying information from it). The second is from validation and service creation and concern product and service catalogue enrichment, before the third supply from operations for instantiation and usage detection (Fig. 61).

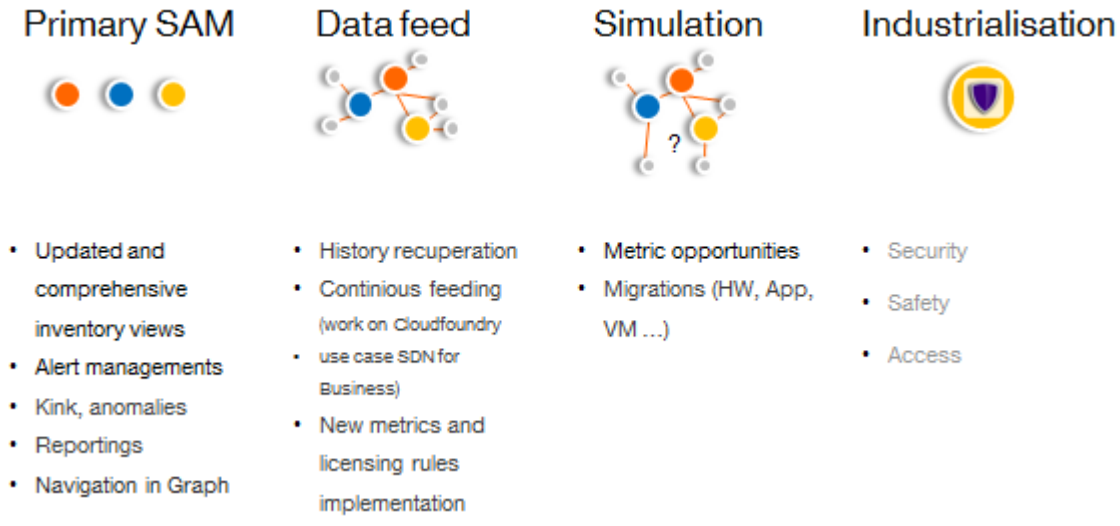

Figure 60 - cSAM tool Features

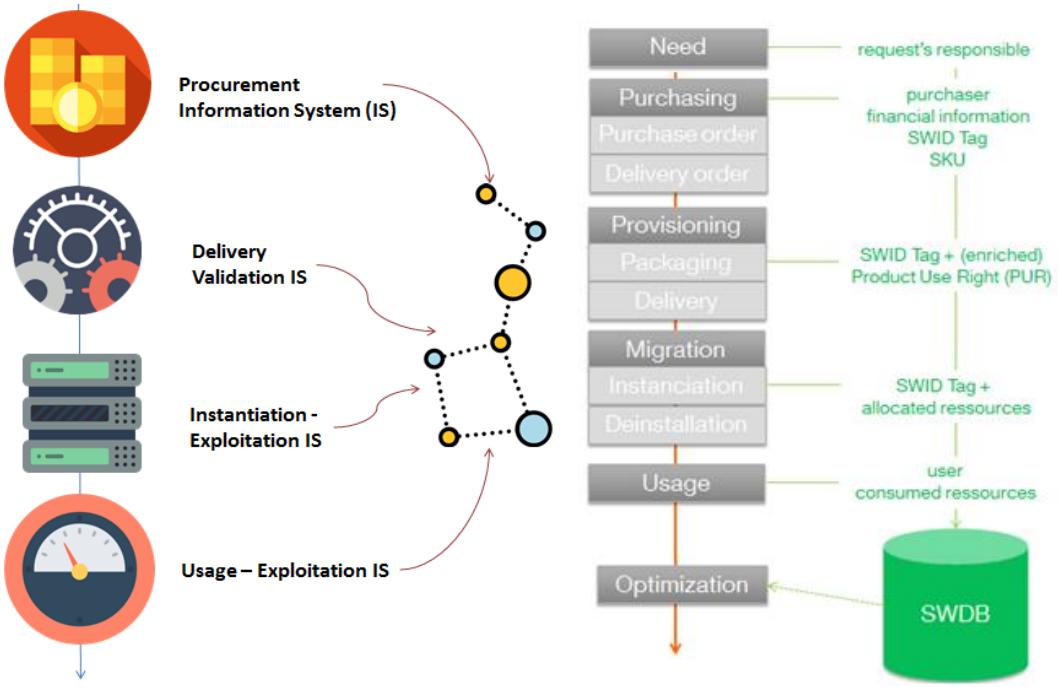

Figure 61 - Asynchronous feeding of graph for Software lifecycle

Thanks to it graph database, cSAM can analyze complex, connected data in easier way than a relational database and persist that analysis for future reference. Depth and density of connections and data volume affect query times importantly. Query data with a depth of millions or tens of millions of connections per second per computer core would be the equivalent in a relational database of millions of "join" operations per second per core, which cannot be done. There is a significant speed difference, and it increases the tighter connections are and the more data you have. The more data you have, the slower it is to link data in other kinds of databases. Using Neo4j, a shortest path query on data with tens of billions of nodes and relationships might take one or two milliseconds to run. The equivalent SQL query would run many thousands of times slower if an application was solely using a relational database.

In terms of simulation, it allows us to simulate a change in licensing model – like metric. In other words, for given software, in a given environment, cSAM can not only check compliancy with contractual conditions, but also simulate other licensing conditions and evaluate their costs in similar technical conditions in order to identify optimal way of licensing (based on usage, on capacity, or on all other criteria like shown on Fig.39)). This knowledge allows orienting potential contract

renegotiations, technical implementation decision, project or service architecture etc.

Fig.62 below illustrates a simulation on our Oracle Database deployment cost evaluation. Currently, all instances of our Oracle DB are licensed under a "Processor"[18] metric. This rule is implemented in a rule engine and running it knowing the current running instances can guarantee compliance with contractual acquired rights. However, it is important to check relevance of such contractual metric considering the current deployments. cSAM allows finding an optimal metric by applying other existing rules in the rule engine or creating a new one on the fly. Thus, in this example, we can easily simulate a metric change, based on real time deployment state and identify that considering our current situation, a Name User[19] metric would cost less, by far (1 453 460€ saved). This metric is also proposed by Oracle, but any kind of metric can be tested included those not proposed by the supplier. Prices used to evaluate the licensing costs might be:

- extracted from observed prices for identical couple of product/metric
- estimated by analogy with a product from same category/metric
- defined via a price list during simulation

[18] Processor: shall be defined as all processors where the Oracle programs are installed and/or running. Programs licensed on a processor basis may be accessed by your internal users (including agents and contractors) and by your third party users. The number of required licenses shall be determined by multiplying the total number of cores of the processor by a core processor licensing factor specified on the Oracle Processor Core Factor Table which can be accessed at http://oracle.com/contracts. All cores on all multicore chips for each licensed program are to be aggregated before multiplying by the appropriate core processor licensing factor and all fractions of a number are to be rounded up to the next whole number. When licensing Oracle programs with Standard Edition One or Standard Edition in the product name, a processor is counted equivalent to an occupied socket; however, in the case of multi-chip modules, each chip in the multi-chip module is counted as one occupied socket."

[19] Name User Plus is defined as an Individual authorized by you to use the programs which are installed on a single server or multiple servers, regardless of whether the individual is actively using the programs at any given time. A non-human operated device will be counted as a named user plus in addition to all individuals authorized to use the programs, if such devices can access the programs. If multiplexing hardware or software (e.g., a TP monitor or a web server product) is used, this number must be measured at the multiplexing front end. Automated batching of data from computer to computer is permitted. You are responsible for ensuring that the named user plus per processor minimums are maintained for the programs contained in the user minimum table in the licensing rules section; the minimums table provides for the minimum number of named users plus required and all actual users must be licensed. Source http://oracle.com/contracts.

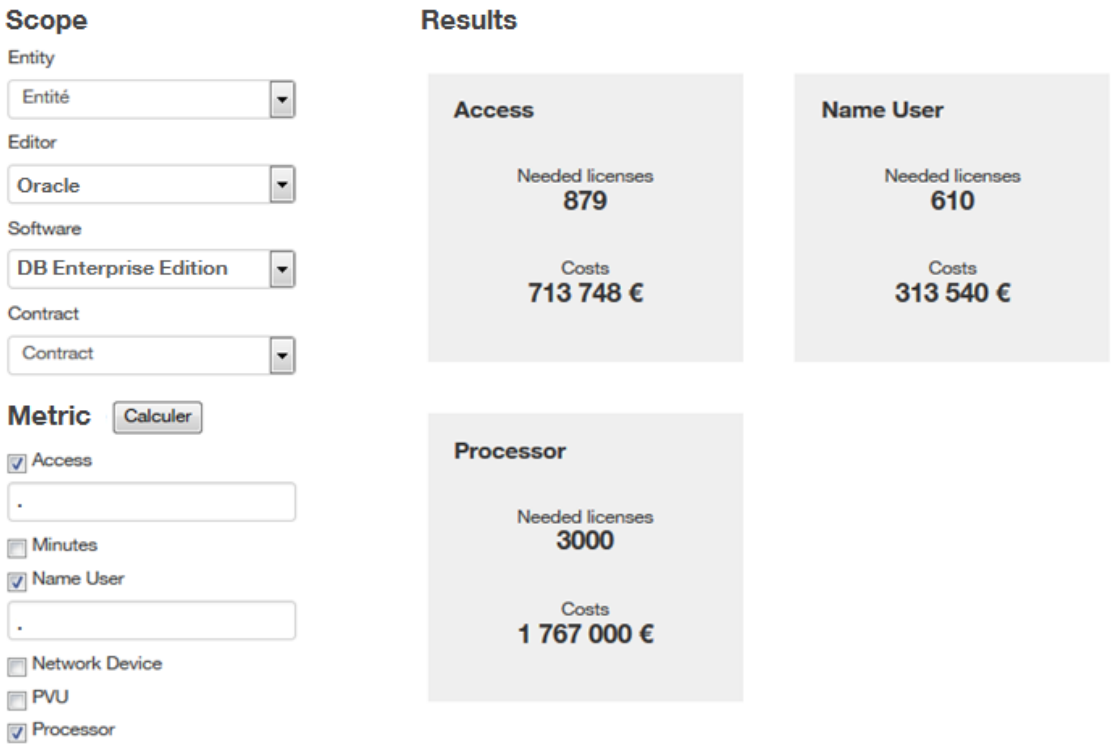

Figure 62- cSAM screenshot - Simulation on Oracle DB licensing

The Fig. 63 illustrates another simulation based on a second scenario: if resources allocated to support an Oracle DB instance will change, we need to evaluate in real time this impact in terms of compliance and costs whatever the complexity of the virtualization architecture. Reminding the definition of the Processor metric giving on previous page, we assume that changing a processor of a physical server will have an impact on all instances bound to it (by analogy, instances under a metric indexed on bandwidth might be impacted by changing the network board). The tool (via a simple graph request ()-[:RUNS*]->()) allows identifying all the impacted instances. In our example, we identify among others, a gap of 28 Oracle processor licenses, which represent an estimated readjustment cost of 27 832€.

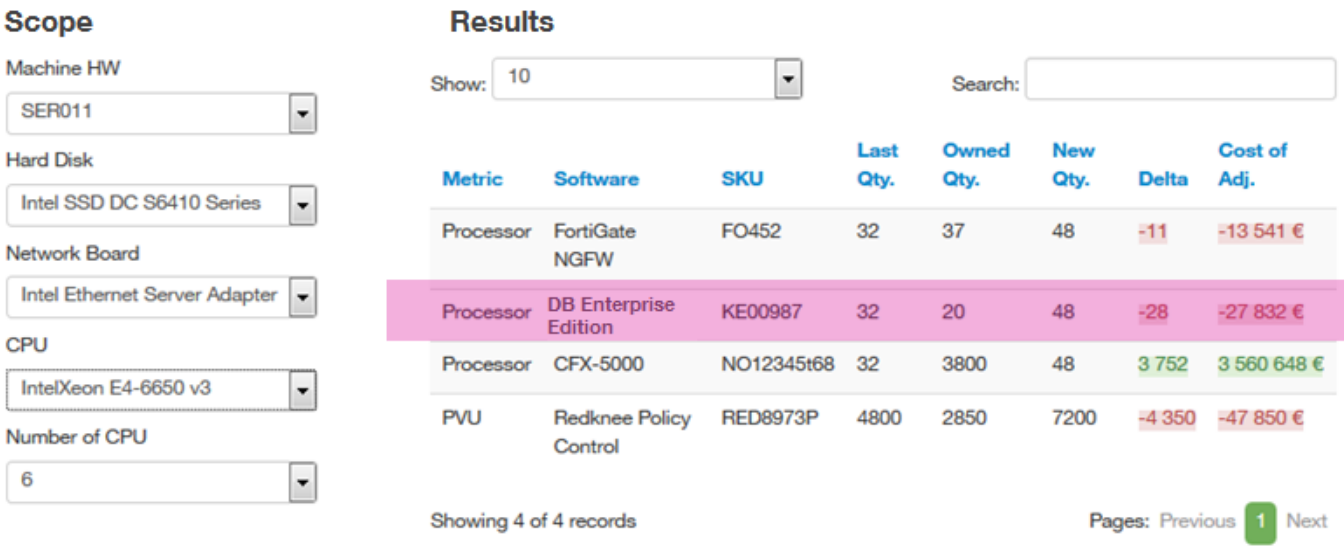

Figure 63 - cSAM screenshot - Simulation on Oracle DB instance's resources

2. Platform Use Case II: Network Function Virtualization

Network functions virtualization (NFV) is a network architecture concept that uses the technologies of IT virtualization to virtualize entire classes of network node functions into building blocks that may connect, or chain together, to create communication services. NFV relies upon, but differs from, traditional server-virtualization techniques, such as those used in enterprise IT. A virtualized network function, or VNF, may consist of one or more virtual machines running different software and processes, on top of standard high-volume servers, switches and storage devices, or even cloud computing infrastructure, instead of having custom hardware appliances for each network function. For example, a virtual session border controller could be deployed to protect a network without the typical cost and complexity of obtaining and installing physical network protection units.

2.1. Orchestration & Hypervisor in Operator's network

With NFV, like in Fig.62, network operators (in other words : telco) are reducing their reliance on single-purpose appliances by taking functions that were previously built into hardware and implementing them in software that runs on industry-standard servers, network, and storage platforms. Beyond reducing network operators' dependency on dedicated hardware, leveraging NFV enables

more programmability in the network and greatly reduces the complexity and time-to-market associated with introducing new services.

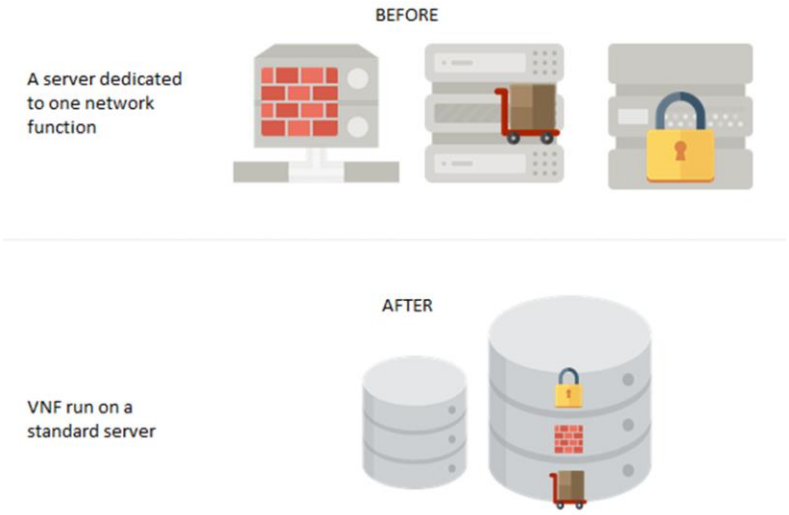

Figure 64 - HW and SW disconnection and separate lifecycle management

Telco need to quickly introduce, automate, and operationalize new virtualized services between data centers, on top of existing network services. Legacy hardware-based appliances dedicated to performing a single function within the network are expensive and wasting resources. Orchestration can resolve this issue by delivering essential framework, templates and processes in order to stitch together virtual and physical resources, as well as automating and dynamically configuring these resources across multiple network domains. Network operators will be able to offer their customers dynamic instantiations of cloud resources, like virtual machines, tenant networks, and storage, on-demand through an enterprise portal with the ability to control network (bandwidth-on-demand) and virtual (VM) resource allocation.

The relation between network software vendors and service providers is deeply changing due to a confluence of economic, market, and technological factors (Fig. 63). Software licensing is complex and may become a hindrance to the adoption of new transformative technology. In such context both service providers and network software vendors would be well advised to bet on trustworthy partnership, promoting emergence of Software Asset Management. The problem has many dimensions, but they sum up this fact: software licenses are overly strict. A license entitles using software in a very specific manner, but many of the licensing schemes in use are not flexible enough to really support the dynamism of

NFV cloud. Inflexible license might inhibit the growth of NFV, because a strict license conflicts with dynamic requirements. No vendor is enough set up to support dynamic entitlements (ie, if you need to scale up an application to meet peak demand without having license in stock). Nevertheless network services on Amazon Web Services acquired through the Amazon Marketplace can be purchased on demand for periods as short as one hour, so dynamic licensing is possible if the proper entitlement infrastructure is in place.

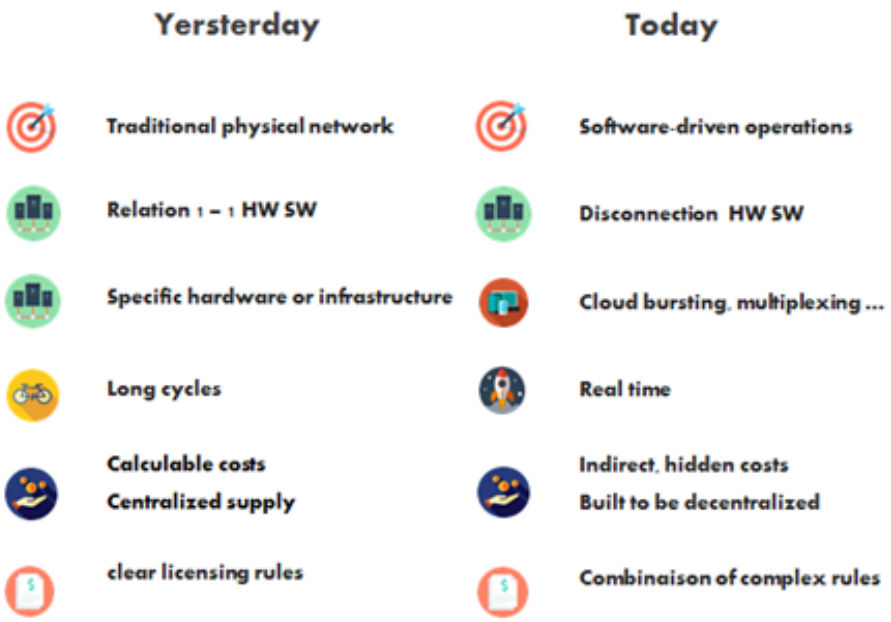

Figure 65 - NFV complexity factors for SAM

NFV architecture separates software purchase decisions from hardware decisions by splitting closed appliances into separate hardware and software components, enabling independent selection of each. Until now, service providers had almost exclusive relations with hardware big vendors (licensing based on invariants such as chassis ID, etc.). They have been accustomed to this sort of comfortable situation. First steps towards NFV force them to take ownership of their own stack. Temptations exist to keep old habits instead of starting a new NFV initiative which will probably cost more than promoting dedicated resource management process. Many service providers have deployed Proof of Concepts (PoCs) use cases in this network function virtualization software but few have the all needed operational tools in place to orchestrate and manage VNF from multiple vendors.

ETSI MANO standards and Open Source initiatives (i.e., OPNVF, OpenMANO, and ONAP) will help service providers in moving toward real implementations. (Open Source Mano, 2017)[34], (The Linux Foundation, 2017)[35], (OPNFV, 2016)[36].

Nevertheless, nothing is easy and complexities of licensing have to be addressed specifically: while service providers and VNF suppliers have different interests to defend in this aspect the value creation for each of them is generated from their collaboration and interdependency. The firsts want to pay as little as possible and only for what they are using, only when they are using it, with the smallest impact on VNF-onboarding process and no service disruption. The seconds need to plan their business and claim they have to protect intellectual property rights (IPR).

Basically, Service providers have interest to promote a usage-based licensing (habitual model in IT), in other words, licensing models with fees that vary with uses. “Use” encompasses notions like time, bandwidth, packets, peaks, etc. Convergence with IT is clearly displayed by the emergence of new players that come with open source “DNA” and open source business model but also with IT inspired business models. Era of single vendor delivering turnkey solution is over and like in IT, service providers needs to integrate new technologies from different vendors.

Main VNF supplier’s concerns are about Intellectual Property Rights (IPR) protection (1) and revenue recognition (2). For this reasons, few of them proposes services/application like integrated license manager or capacity tracking manager; in order to report application uses and aggregate it, decide if it licensed to run, can be cloned, etc.

(1) Licensing must meet service provider requirement while being easy to implement but preventing unauthorized use of the software. Network functions are virtualized and may run on different host hardware at different times, i.e. elastic scaling and be easily cloned as part of regular operation like migration/backup but enable rogue employee or attacker running stolen software. Vendors want to prevent misuse to secure their IPR, but it comes with inconvenience: too much protection could be too inconvenient to use (i.e. service interference, legitimation of VM cloning, tie to specific hosts, extension to future applications, etc.). It implies that the responsibility of the license compliance fall back on Software vendors; just the same, usage monitoring and control.

(2) VNF vendors propose to connect their license manager to business system to be able to recognize what to bill and consider as revenue (Fig.64). It

questions about the vendor usage supervision legitimacy and might convey a business encroachment to the cost of service providers.

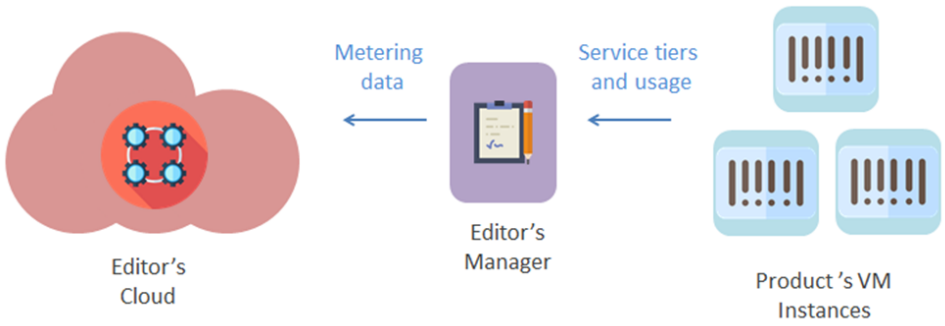

Figure 66 - License & metering server management model

The fact is that since years IT Software is mainly distributed on "declarative license" mode. In other words, during contracting phase, Software supplier trusts Software buyer and adjust negotiated license quantity on the amount of licenses that will be installed. Software installation and usage do not required interaction with any license manager because IPR protection is guaranteed by first clause of contract signed between Software vendors:

"This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited."

This clause quoted from a standard End-User License Agreement (EULA) proposed by (Oracle, 2010)[37], is nearly the same than clauses proposed by other well-known IT software vendors. These contracts are often jointly proposed with "True-up" process (Microsoft, 2017)[38]: an annual reconciliation process through with you can increase or decrease your license subscription counts. Main benefit from this system is that customer keeps controls on what, where, when and how he deploys Software, processes his own allocated/consumed resource and asset optimization. It is translated into usage-based metrics like presented on Fig (usage).

Hard truth is that while NFV offers stronger partnership opportunities between service providers and Software vendors, first contracting methods do not reflect expected trust between partners. Trust is not a matter of technique, tricks or tools but of character and will.

Considering experience and process maturity on IT level, relation with software editors based on declarative license uses and perpetual usage rights seems to be the best approach to follow. Our aim is to replicate relevant software IT processes on production optimization as much as possible when relevant. VNF vendors can allow tremendous innovation and growth to telco industry, on condition that related software licensing is adapted to the service providers and do not stand in the way of fast on-boarding of VNF.

Fact is that trust is built with consistency so : to turn into declarative license uses and perpetual usage rights, service providers need to have generic and reliable process and tools to demonstrate their audit-readiness and accurate counting loop. It involves setting up relevant SAM program which will first address the prerequisites developed in chapter 3.

2.2. CONTEXT CONCERNS

As a reminder, a NFV cloud is designed to host and deploy several virtual network functions (VNFs) using a cloud network. Before, to deploy a firewall service or a Wide Area Network (WAN) service, operator were installing specific customer premises equipment (CPE) to deliver the service. Using an NFV model, telco can deploy NFV servers in data centers, and then deploy VNFs and network services to the customer using software. One of the NFV advantage is that carriers can significantly reduce new services 'time to market' and software rather than specialized hardware networks.

Blue Planet delivers carrier-grade NFV orchestration capabilities for instantiating, managing, and chaining Virtual Network Functions (VNFs). Blue Planet provides Management and Orchestration (MANO) guidelines to manage and automate the VNF lifecycle, and intelligently orchestrate NFV Infrastructure (NFVI) resources across multiple data centers. It leverages RESTful APIs and model-driven templates to simplify integration with different OSS, SDN, and Virtual Infrastructure Manager (VIM) platforms. Blue Planet is cloud management plateform neutral: OpenStack and VMware are supported today, and architecture allows support of other cloud management alternatives. The Fig.65 explains the context the architecture of our experimental environment. The aim is to provide a Network as a Service based on an open and flexible NFV WAN strategy.

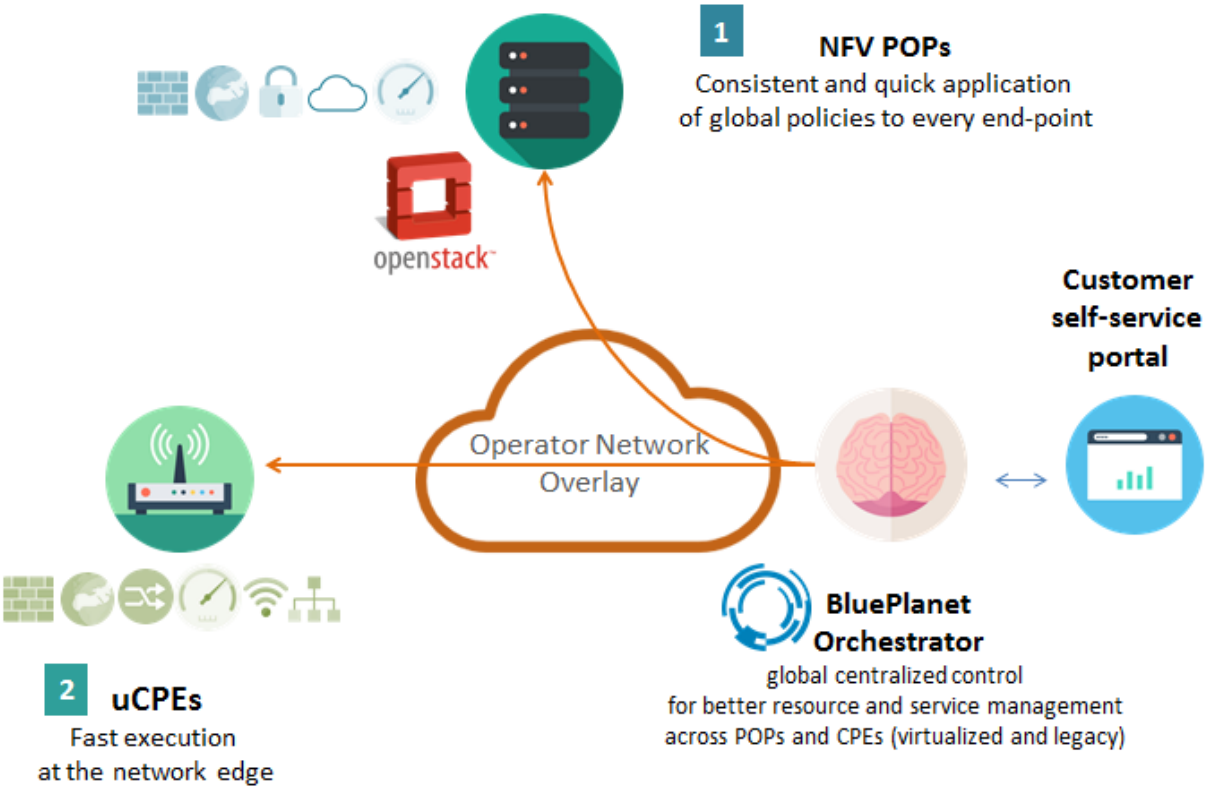

Figure 67 - NFV Cloud Orchestration

The Blue Planet User Interface (UI) yields a real-time view, dynamically updated to show changes in the network service or the addition/removal of a VNF from the service chain (Figure below illustrates it). Moreover, we can move through hierarchy on various elements to get more details on individual VNFs and network services and get access to supporting resources.

This use-case is very interesting for carriers' point of view because it deals orchestration allows monitoring of, simultaneously, NFV Point-of-presence which are consuming a lot of resources but in limited amount (less than 15 distributed in Europe) mainly based on Open-stack infrastructure; and universal Customer Premise Equipment (CPE) mainly enterprise clients. uCPE do not consume a lot of resources but will potentially be very numerous (millions of equipment). Concerns related with this two types of equipment (POP and uCPE) will generate different issues like reliability, real-time and volume which should be addressed in a different approaches but through the same operation center.

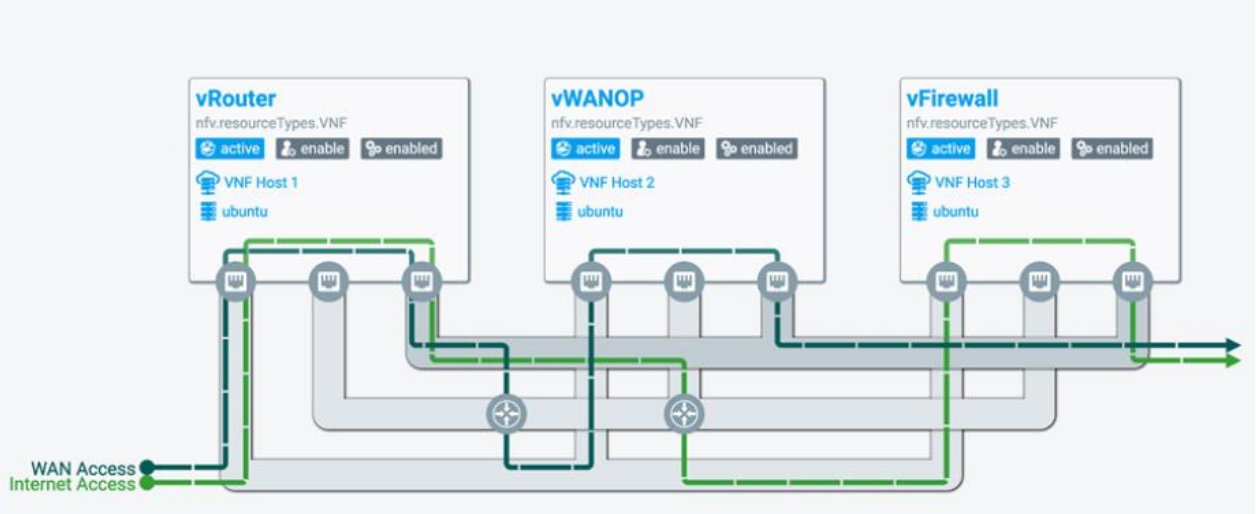

Figure 68 - Blue Planet UI

Like shown on Fig.66, Blue Planet (BP) questions POP's OpenStack instances about deployed virtual machines and related allocated resources. OpenStack returns information like BP ID, VM's creation timestamp, VM's end timestamp, cpu/ram/disk allocation, running image name, admin ip address, a label typical of VM purpose, tenant id and allocated resources and a unique hostname. The serial number, or any information related with license management is reachable through the vendor management module (here FortiManager provided by Fortinet) using the admin ip address as primary key.

To get information directly related to distinguishing features of an instance, a dedicated connector converses with the virtual machine in SSH mode, to recompose a configuration bloc and obtain such information like identification metadata or potential running options. For example: a Firewall instance runs an antivirus function (an option): it is visible in the configuration bloc, found by key-words research like "antivirus" and "enable". A connector depends on product vendor and is provided either by the vendor itself, the orchestrator or is developed by the service provider. In case of Fortinet, the connector already exists in BluePlanet. Nevertheless, in case the product will go up a version, we will be dependant from the one in use by the connector and might be not able to recognize a running option. In such case, we should implement a module as generic as possible from the connector to be as less as possible dependent on the version to catch VNF's internal configuration.

This allows us to get all configuration data about an instance (in other words configuration data) like VM sizing. Linked with the figure above (usage), we get access to allocation (allocated resources like CPU, RAM, disk, sockets, bandwidth, etc…) and supervision like network sizing, link between instances, managed objects etc… We can easily get running duration of each network service using timestamp function attached to each instance. Nevertheless, BluePlanet does not provide yet consumption usage like traffic (event or flow per seconds, data or bandwidth), amount of access (direct, not direct) or calls, neither resource real consumption (like physical CPU consumption per minute). As standards assumes that one VNF only can be instantiated on one VM (or more) we make assumption that we can get this information through OpenStack using program like Telemetry. Yet, we did not find opportunity to implement it but Telemetry's aims are to collect reliable data on the physical and virtual resource usages comprising deployed clouds, to persist these data for subsequent retrieval and analysis, and trigger actions when defined criteria are met the Telemetry requirements of an OpenStack environment are vast and varied, they include, among other, use cases like metering, monitoring, and alarming.

2.3. USAGE COLLECTION

The Telemetry Data Collection services can efficiently polls metering data related to OpenStack services; collects event and metering data by monitoring notifications sent from services and publishes collected data to various targets including data stores and message queues. The Telemetry includes the following components:

- A compute agent *(ceilometer-agent-compute)* which runs on each compute node and polls for resource utilization statistics.
- A central agent *(ceilometer-agent-central)* which runs on a central management server to poll for resource utilization statistics for resources not tied to instances or compute nodes. Multiple agents can be started to scale service horizontally.
- A notification agent *(ceilometer-agent-notification)* which runs on a central management server and consumes messages from the message queue to build event and metering data. Data is then published to defined targets.

2.4. MODELING

Our objective is to validate the fact that our graph model can be managed in NFV cloud environment (Blue Planet Orchestration/OpenStack/uCPE). It will validate our assumption that the model can be used to easily model complex

platforms and software. To achieve these experiences, we consider a Firewall VNF from Fortinet.

We choose the Fortinet virtual Firewall (FortiGate VMX) example for several reasons:

- FortiGate VMX licenses can be defined by several types of entitlements and performance values vary depending on system configuration
- It will allow us to increase complexity of our use case such as: integrating controls between product's link (FortiGate VMX can be enriched by options) and constraints of uses (based on technical specifications and system performances).

To evaluate relevance of our model, we propose to define in Fig. 69 below a NFV architecture model.

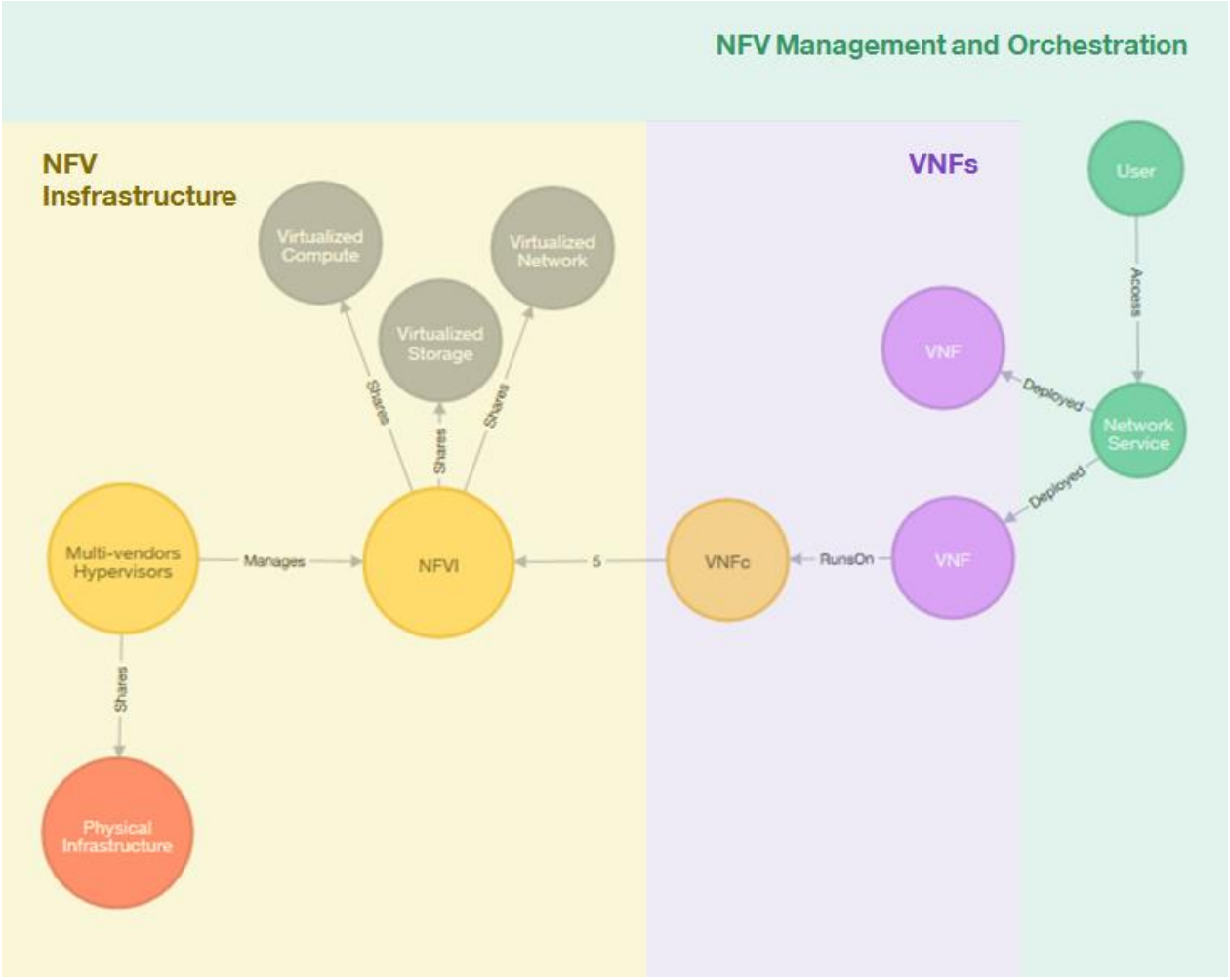

Figure 69 - NFV Architecture Model

The diagram illustrates:

VNFs represents the collection of Virtualized Network Functions: a Service Provider implements network services using VNF instances (which shall encompass several software components called VNFc) running on common infrastructure elements.

The NFV Infrastructure (NFVI) depicting the mapping (virtualizing) of physical servers and network facilities onto equivalent virtual functions. The NFVI shall provide compute capabilities comparable to an IaaS cloud computing service as a run time execution environment as well as support the dynamic network connectivity services. The computing nodes of the NFV Infrastructure will be located in NFVI-Points of Presence (PoP) or embedded in other network equipments. The resource pooling concept includes a notion of multi-tenancy -

where the same pool of resources supports multiple applications from different administrative or trust domains.

The NFV management plane, with various independent VNFs all competing for resources, the management plane is responsible for allocation of the physical resources in a fair manner to support various Service Level Agreements.

a. Graph model construction

In this section, we will follow the Software lifecycle proposed in Section IV and refer to the Fig.2 about SAM maturity scale.

i. Purchasing

Fig.70 might be an extract from VNF Market Place which proposes the product we identified as needed and are planning to buy under a specific metric 'Instance'.

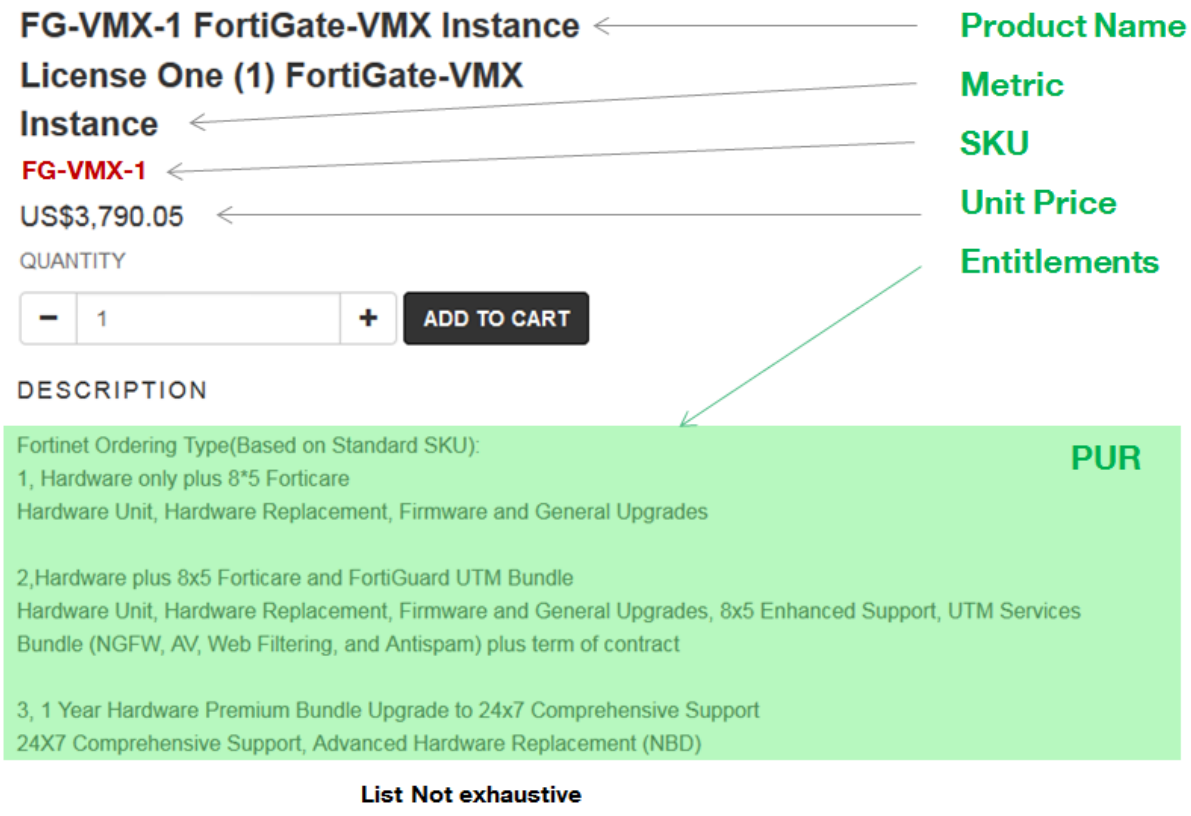

Figure 70 - Product Catalog (2)

Few elements (in green above) are necessary to identify precisely this offer and determine the level of grants (PUR) given by this type of licensing. These

elements have to be collected in the purchase order and reconciliated with data from the delivery order. We can notice in this offer that softare's editor is missing. Some research is necessary to identify Fortinet.

In the graph (Fig. 71): first step is to create our product, with a label 'VNF' and several attributes found in the purchase order. In the same way, we create a label 'Supplier' and 'Editor' to identify a node 'License Store' and 'Fortinet':

Figure 71 - Neo4J interface – Graph 2 Step 1

```
CREATE (S:VNF {NAME:"FORTIGATE-VMX", SKU:"GF-VMX-1",VERSION:"5.4", CATEGORY:"FIREWALL"})

MERGE (R:SUPPLIER {NAME:"LICENSE STORE"})

CREATE (E:EDITOR {NAME:"FORTINET"})
```

Then, in Fig. 72, we create several nodes with label 'PUR', which represents scope of usage, metrics, environments … The following list is not exhaustive regarding current Fortinet definition of this couple product/metric.

```
CREATE (P:PUR {TYPE:"METRIC",METRIC:"INSTANCE"})
```

```
CREATE (P2:PUR {TYPE:"PERFORMANCE",CONCURRENTSESSION:"NO LIMIT"})

CREATE (P3:PUR {TYPE:"SPECIFICATION ", VIRTUALDOMAINSMAX:250})

CREATE (P4:PUR {TYPE:"SPECIFICATION ", USERLICENSE:"UNLIMITED"})
```

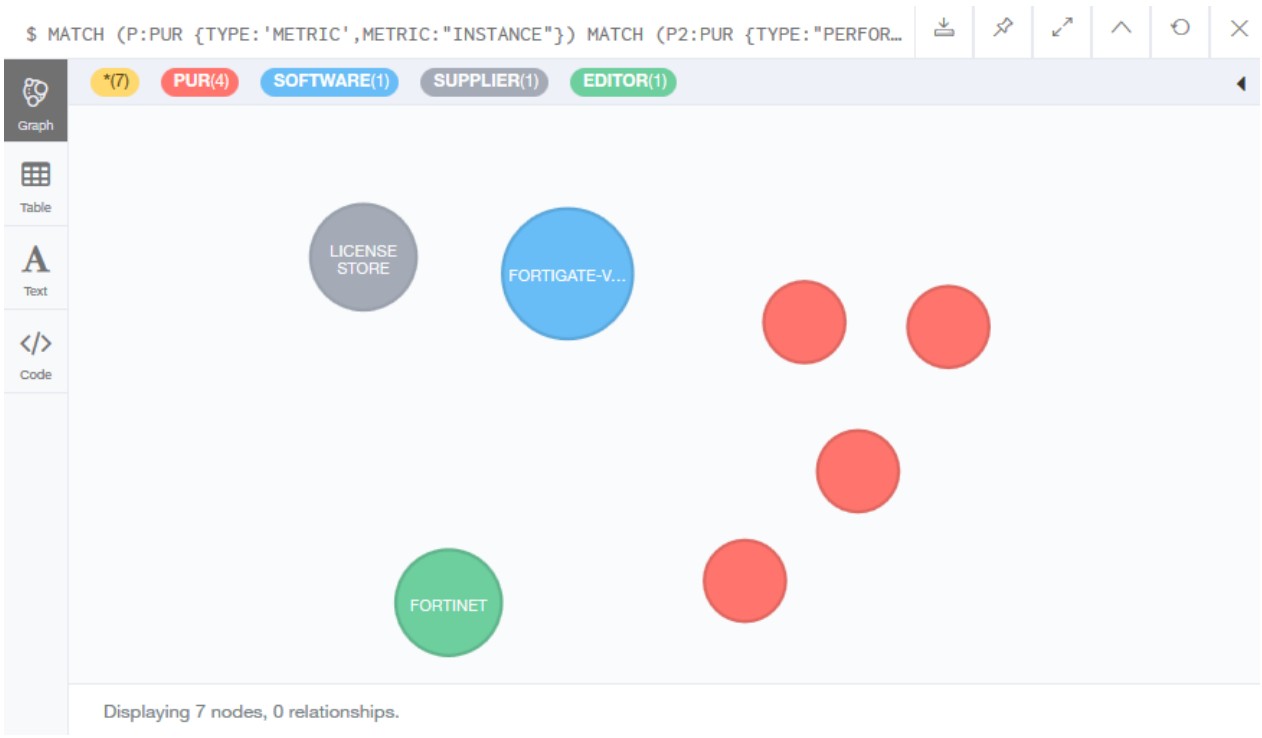

Figure 72 - Neo4J interface – Graph 2 Step 2

Then, in Fig. 73: to create relations between nodes:

- Between an editor and VNF (EDITS): 'Fortinet' edits 'Fortigate-VMX'
- Between a VNF and PUR (DEFINES): 'Fortigate-VMX' is licensed under Instance metric
- Between a supplier and a product (DISTRIBUTES): 'License Store' distributes 'Fortigate-VMX'.

Once again, this process and collect are essential to fulfill Identification requirements: PUR are translated in the SKU, this SKU enriches the SWIDTag delivered during provisioning processes; it guarantees the link between a contract and VNF/ VNF and Instance.

```
MATCH (P:PUR {TYPE:'METRIC',METRIC:"INSTANCE"})
```

```
MATCH (P2:PUR {TYPE:"PERFORMANCE",CONCURRENTSESSION:'NO LIMIT'})

MATCH (P3:PUR {TYPE:"SPECIFICATION ", VIRTUALDOMAINSMAX:250})

MATCH (P4:PUR {TYPE:"SPECIFICATION ", USERLICENSE:'UNLIMITED'})

MATCH (S:VNF {NAME:"FORTIGATE-VMX"})

MATCH (R:SUPPLIER) MATCH (E:EDITOR {NAME:"FORTINET"})

MERGE (C:CONTRACT {NAME:"CUIMT002",DATE:18-10-2017,CONTACT:"ZOE"})

MERGE (SP:ENTITY {NAME:"SERVICE PROVIDER"})

MERGE (P)-[D:DEFINES]->(S) MERGE (P2)-[D2:DEFINES]->(S)

MERGE (P3)-[D3:DEFINES]->(S) MERGE (P4)-[D4:DEFINES]->(S)

MERGE (E)-[E1:EDITS]->(S) MERGE (R)-[D1:DISTRIBUTES]->(S)

MERGE (R)-[S1:SIGNS]->(C)  MERGE (SP)-[S2:SIGNS]->(C)

MERGE (C)-[DC:DEFINES {QUANTITY:2, UNITPRICE:"3790", CURRANCY:"$"}]->(S)
```

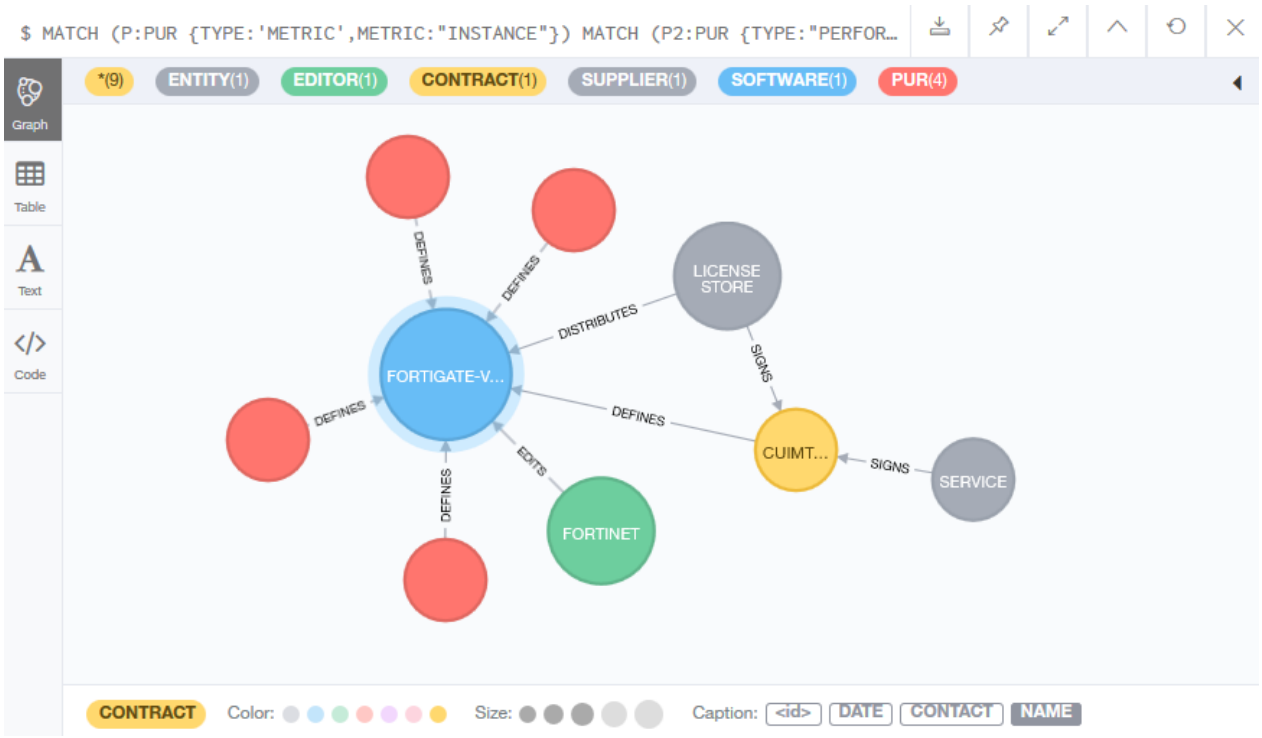

Figure 73 - Neo4J interface – Graph 2 Step 3

ii. Provisioning

We create a label 'NetworkService' and a node 'NetworkService1' which will include our Firewall. Each NetworkService has its own SKU characterizing its composition and eventual licensing models (in case where the service provider will propose it as a commercial offer).

The relations 'CONTAINS' is enriched by properties like a service id (specific SKU), in Fig.74.

```
MATCH (S:VNF)

CREATE (A:NETWORKSERVICE {NAME :'NETWORKSERVICE1', SKU:"NS001"})

CREATE (A)-[C:CONTAINS]->(S)
```

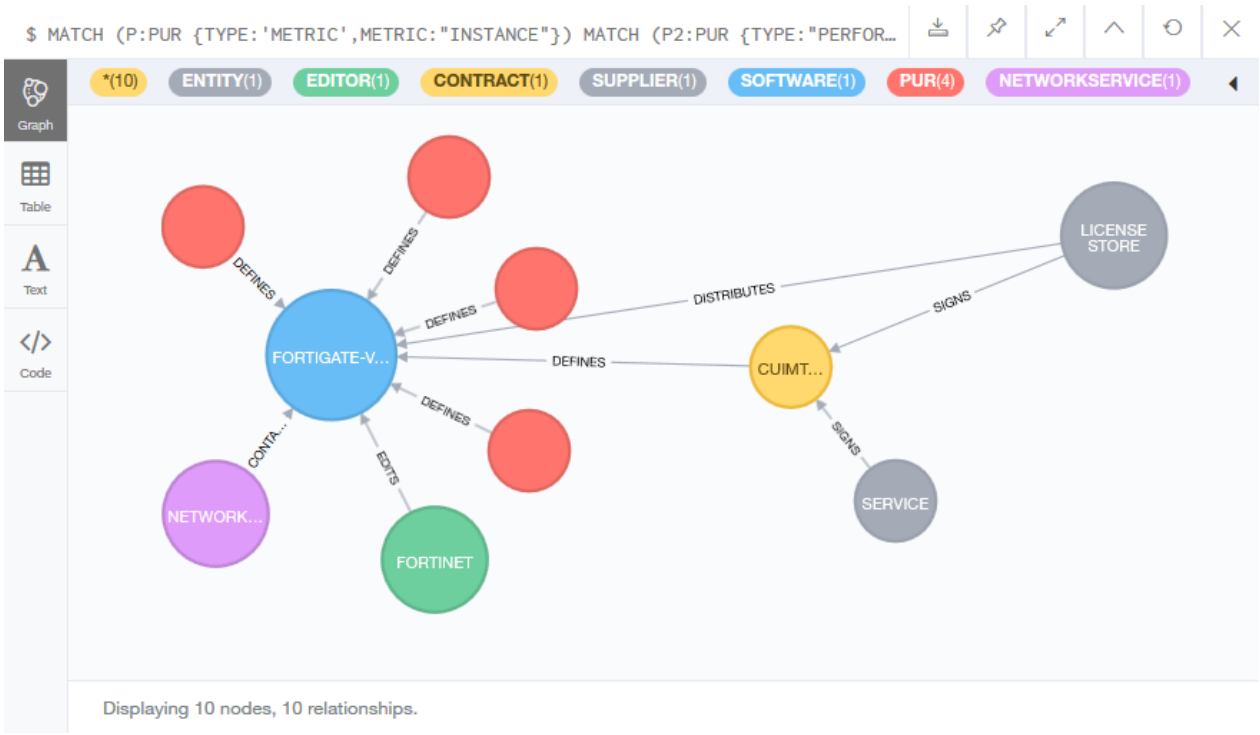

Figure 74 - Neo4J interface – Graph 2 Step 4

iii. Instantiation

To fulfill the step 1 (*visibility*) of the maturity scale, we need to have an exhaustive view of infrastructure, resources and instantiation.

Crucial point is now to create a link between the instance and the product which we bought (Fig. 75). The Orchestrator knows and updates all identification elements of it components. This allows creating the link between the product in catalogue and the installed product.

```
MATCH (S:VNF{NAME:'FORTIGATE-VMX'})

MERGE (I:INSTANCE {NAME:"FW-VM1",SKU:"GF-VMX-1",IMAGE:"FORTIGATEVMX1"})

CREATE (I)-[I1: INSTANTIATES] ->(S)
```

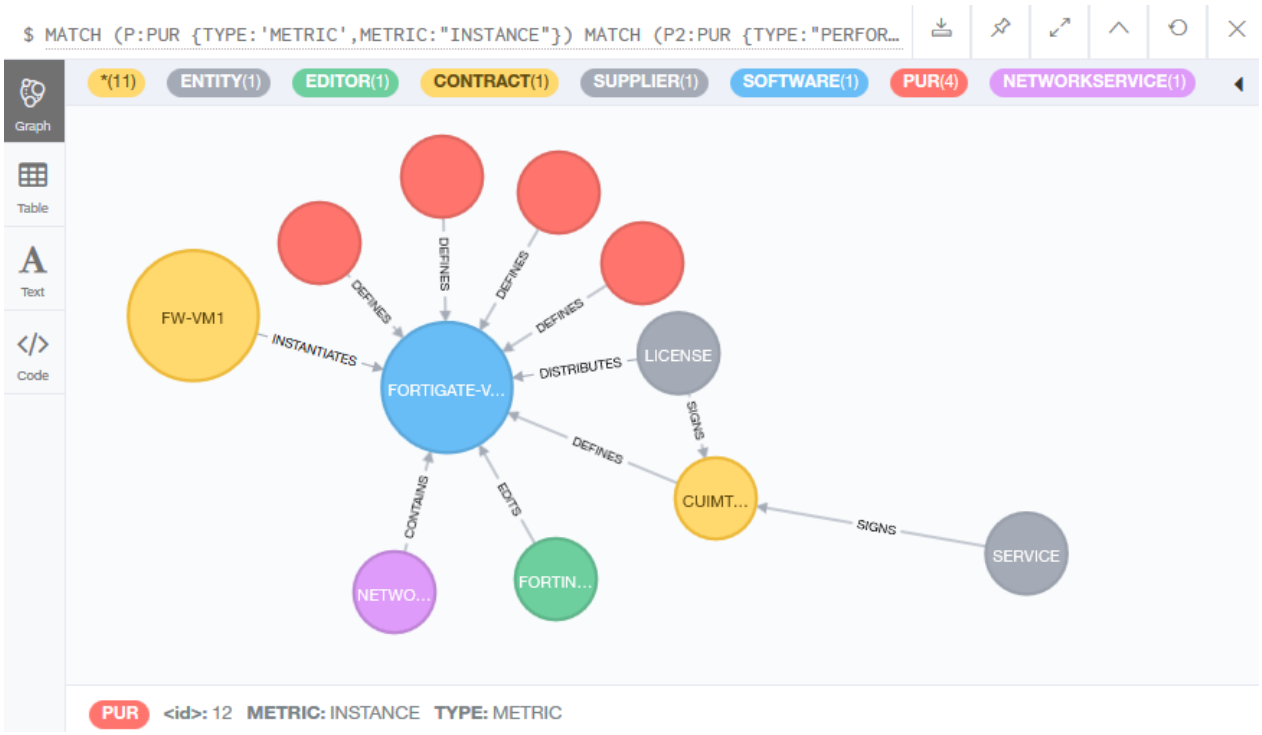

Figure 75 - Neo4J interface – Graph 2 Step 5

The characteristics of the VNF, its instances and their configuration have to be traceable regardless of the deployment conditions. This requirement of traceability encompasses precise identification of Software and resources to allow maintenance of deployment inventories. IaaS (Open Stack) offers bare HW resources access which are visible through Blue Planet monitoring interface.

In our example, the firewall has been deployed on the cloud via the orchestrator and run as one instance (Fig.76).

```
MATCH (I:INSTANCE {NAME:"FW-VM1"})
MERGE (VM:VM {NAME:'VM2',CPU:4, RAM:8})
MERGE (T:TENANT {NAME:'TENANT2',NBINSTANCE :20, RAM :100})
MERGE (V :VLAYER {NAME:'OPENSTACK2', REGION:'FRANCE', VERSION:'ZOE'})
MERGE (M:MACHINE {NAME:'BAREMETAL2'})
MERGE (R:RESOURCE {TYPE:'CPU2', RAM:'X86'})
CREATE (VM)-[R0:RUNS]->(I)
CREATE (T)-[R1:RUNS]->(VM)
CREATE (V)-[R2:RUNS]->(T)
CREATE (M)-[R3:RUNS]->(V)
CREATE (M)-[R4:HAS {NUMBER:20}]->(R)
```

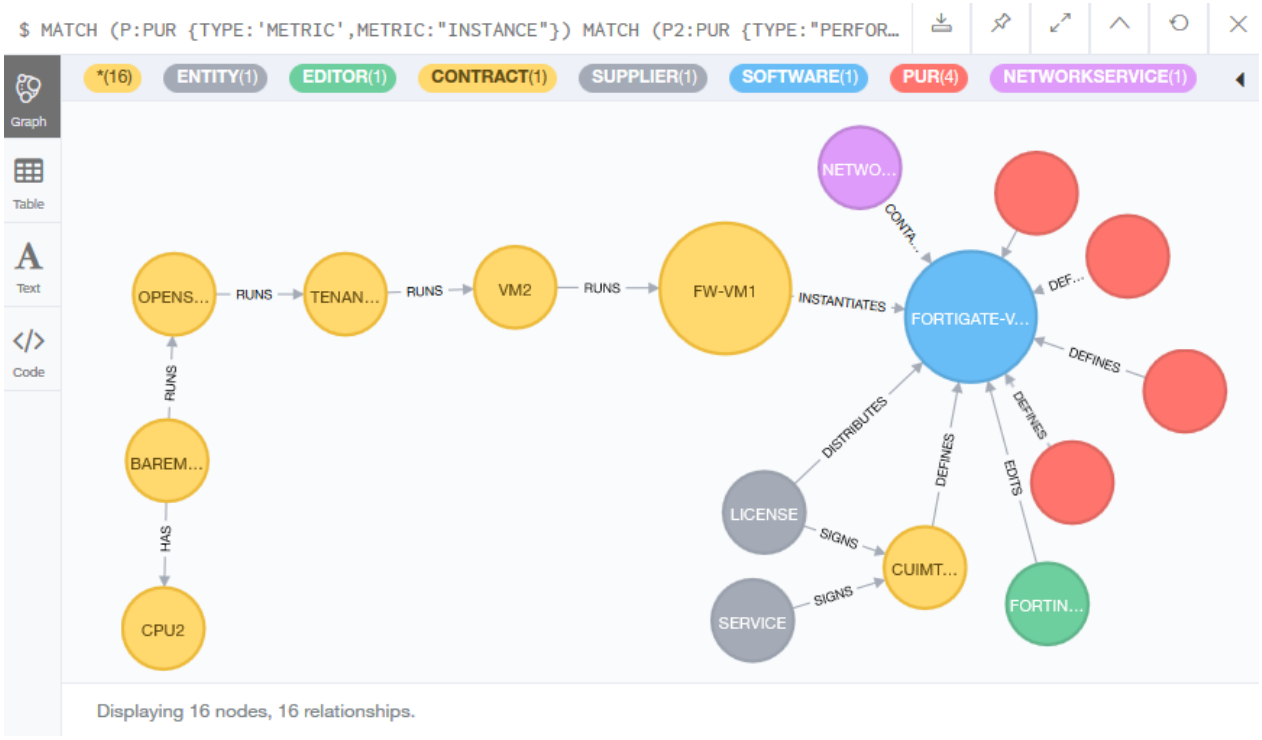

Figure 76 - Neo4J interface – Graph 2 Step 6

In case of migration or backup purpose, among others, we need to keep and maintain a link between instances. We propose to create a relation between

iv. Options

There are a number of features in our Firewall that can be configured to either be displayed or disabled. Activation of feature influences the licensing conditions and prices. We choose to represent it using a label 'Feature', each node 'Feature' represent a function of the Firewall. The relation between the instance and the function is enriched by the status of activation. '1' represents a displayed feature, '0' represents a disabled one (in Fig.77).

```
CREATE (F:FEATURE {NAME:"CENTRAL NAT TABLE"})
CREATE (F2:FEATURE {NAME:"LOAD BALANCE"})
CREATE (F3:FEATURE {NAME:"EXPLICIT PROXY"})
CREATE (F4:FEATURE {NAME:"DYNAMIC PROFILE"})
MATCH (I:INSTANCE {NAME:"FW-VM1"})
CREATE (I)-[F5:FEATURES {STATUS:1}]->(F)
CREATE (I)-[F6:FEATURES {STATUS:1}]->(F2)
CREATE (I)-[F7:FEATURES {STATUS:0}]->(F3)
CREATE (I)-[F8:FEATURES {STATUS:1}]->(F4)
```

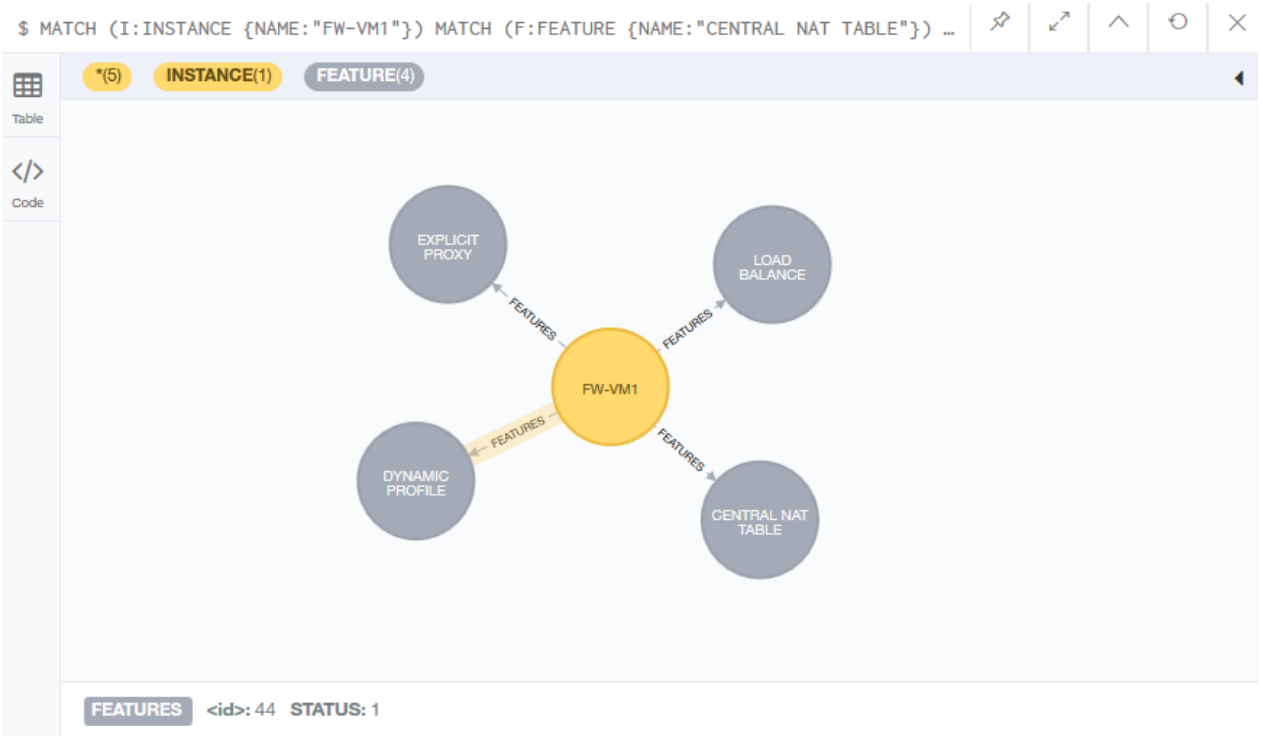

Figure 77 - Neo4J interface – Graph 2 Step 7

b. Cost-saving Identification

Like in section V.1.5.c, we propose two simulations. The first one simulate a metric change for the Firewall Fortigate from Fortinet (Fig.78). Based on relation in the graph including usages, user access, allowed users, time spent on application, running time of all instances, virtual link between instances etc., we easily identify that for this specific software, in this scope (Entity), under Access metric, we would need 879 licenses for an evaluated cost of 186 348€. Editor might not propose all these metrics in his offer, nevertheless this knowledge is valuable. In case were a metric does not exist for a given editor, cSAM proceed by analogy with software and price from the same category. Accuracy of the simulation depends on the volume of metric and software in the base.

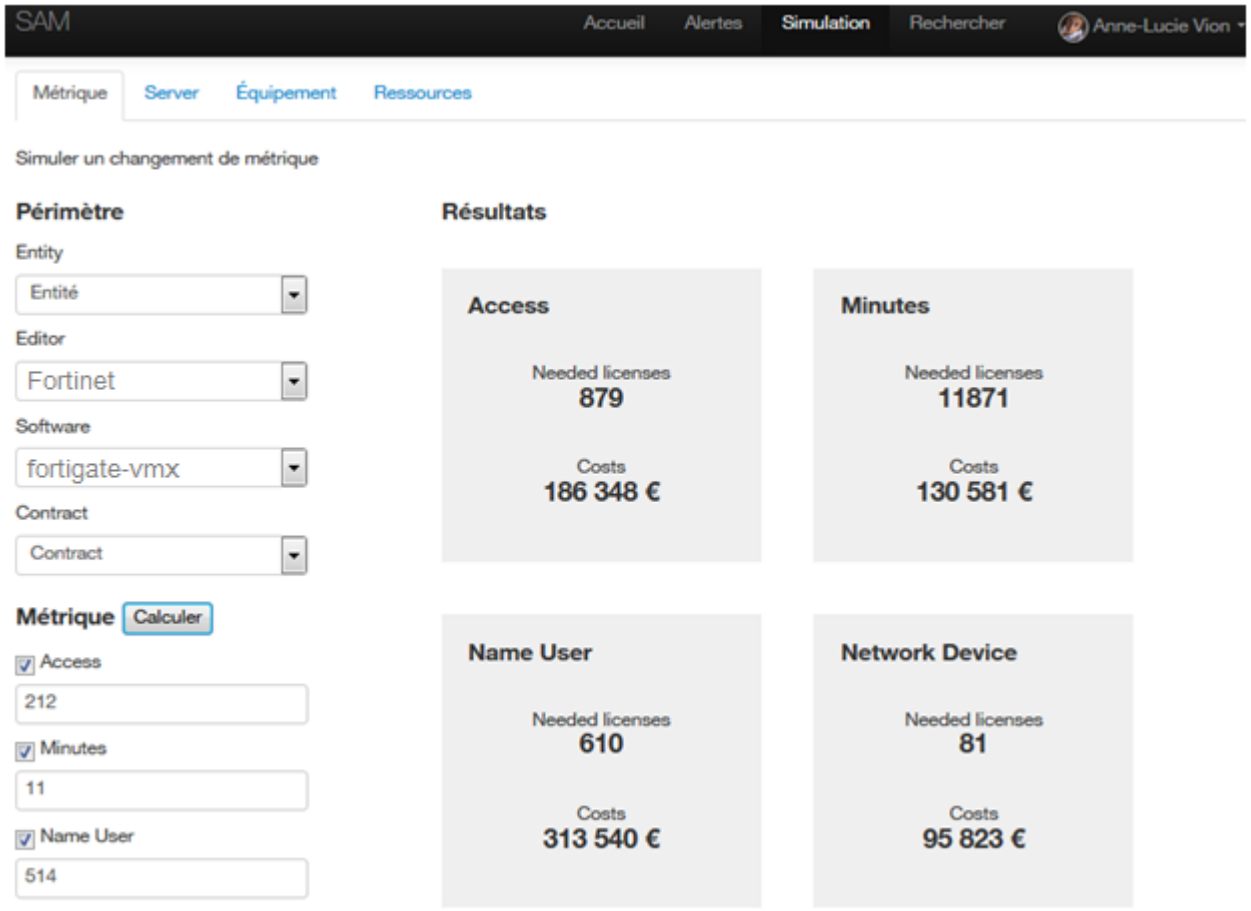

Figure 78 - cSAM - simulate metric change

We can instantly (thanks to the chain or "runs" relations) evaluate potential impact of resources reallocation on each virtual or physical layer(s) in terms of licensing costs. For example, changing a CPU in a data center or a tenant has an impact for all instances with a processor-capacity-based metric; changing network board has an impact on instances with a traffic-based metric.

Fig.79 illustrates a resource change : what will be the impact of a processor replacement on hardware (here the machine SER001)? Graph allows winding up each layers of virtualization and to identify impacted instances (which are supported by this physical and/or virtual equipment given a processor-based metric). cSAM calculates previous amount of needed license, new amount after resource changes, compares it with license stocks and evaluate costs of adjustment (based on contract's average prices). Estimated cost of adjustment is around 75k€. It offers new interesting vision of infrastructure optimization, adding a license criteria.

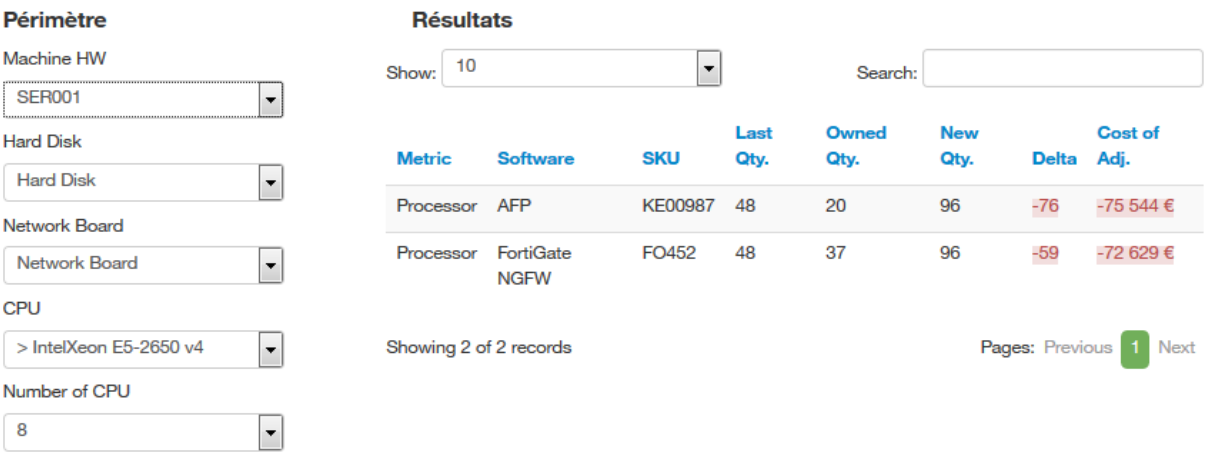

Figure 79 - cSAM - simulate resource change

Chapter 6

VI. CONCLUSION

This final chapter will be organized as follow. (1) We will remind the keys issues who motivated our works; (2) we will remind our main contributions and (3) propose a set of tracks to overcome their current limitations.

1. REMINDING THE ISSUES

Cloud computing represents a more dynamic and flexible approach to provide resources on hardware and software level. It supposes innovative distributed architectures, ownership and controls as well as new software pricing models.

The disruptive influence of cloud computing on software licensing has to be taken into consideration mainly because traditional and complex licensing models often jeopardize using these products in the cloud. The flip side of the flexibility promised by large-scaled virtualization is that software licensing issues may hold back the benefices offered by the Cloud environments. The rise of shelfware and the growing number of license audits by commercial software vendors are together raising awareness of the software license risks, counterfeiting and over-deployment. This context stresses the necessity to adapt and reinforce automation of current SAM processes when organizations use cloud computing.

Moreover, some techno-economic drivers are converging to create a paradigm of change in the design and operation of future telecommunications

networks and services. “Softwarization”, currently impacting the Network, highlights a new dimension of network management. We assume that software license’s management in real-time and on large-scale cloud environment will sophisticate Virtualized Network Function (VNF, or Network Software) on-boarding processes. Network virtualization and softwarization disrupt software licensing business models.

Altogether, the complexity of software lifecycle management, the multiplication of actors in this cycle and the lack of efficient tools, lead to an understandable disconnection between software usages, associated hardware and the related licensing model. Also, because cloud environments tend to automate software lifecycle management, SAM processes are expected to be automated as well. The combination of on premise solutions and modern, fast growing cloud technologies makes it hard to manage the software lifecycle as a part of SAM. Many products provide their own license enforcement mechanism; however, most of these are problematic for use in the cloud. These mechanisms may depend on features difficult to provide in the cloud, such as hardware keys, physical server IDs, CPU class, and global user identity. Where schemes can be implemented on one cloud, they cannot generally span multiple clouds (hybrid model). The main hindrance to overcome this issue lays in identification of software during it all lifecycle, including identification of product uses rights (PUR). Heterogeneity of the restriction and right’s nature make it extremely difficult likewise identification of consumed cloud resources to run software on each.

2. REMINDING THE CONTRIBUTIONS

The idea that we developed is that turning to the Cloud is not changing the object of SAM, but altering how SAM processes should be designed. Turning to the Cloud, SAM controls must be done in real time taking into account the fast rhythm of changes: services are provisioned, configured, reconfigured and decommissioned in a matter of minutes. Compliance risks are increased by the ease and speed of provisioning, which can bypass traditional centralized processes. As software becomes omnipresent, we developed the necessity to adopt existing and relevant software license optimization IT process. Furthermore we questioned the emerging contractual relation trends between service providers and network software editors;

Effective SAM results in the ability to have accurate and complete view of software assets entitlements that are owned, deployed and used. However, if most

of the recognition tools are quite efficient (we proposed an evaluation of the most often-used), a common mistake is to underestimate the process of identifying software after discovery. There is a huge difference between software discovery, software recognition and software management. If ISO/IEC 19770 is currently the most advanced proposition to overcome software identification throughout it whole lifecycle, we underlined it relative efficiency in particular because of software market weak adherence. For efficiency reasons, we proposed to adapt the ISO/IEC 19770-2 with a concept borrowed from large retailers: Stock Keeping Unit. The intended benefits of this better management of identification include easier demonstration of proof of ownership, cost optimization of the use of entitlements and easier license compliance management.

Software Asset Management is mainly about deciding about a strategic approach of understanding software needs so that their deployment's efficiency and effectiveness will contribute to maximize the return on investment. The fact is that license optimization requires a major shift within a company to implement proactive SAM processes and be able to harness the power of this decisive business asset. We proposed a model for SAM approach in the cloud based on control loop and automatic computing concepts; we discussed about relevance of using a graph database as a central process data connection. Then we proposed a qualitative evaluation of our model based on two relevant use-cases. First use-case deals with PaaS Instanciation through Cloud Foundry. Second deals with virtualized network function orchestration. We demonstrated that the model can be adapted to fit complex and distinct cloud environment overtaking concerns brought by complex licensing models.

Following this model, we proposed a multi-domain prototype build to integrate cloud dynamicity issues; designed to integrate quickly and easily new composed metrics support innovative simulation functions to allow better uses and deployment control. The simulation functions encompass possibility to simulate a change in metric and evaluate the best in terms of licensing costs depending the inventoried deployments and uses; and the possibility to evaluate the impact of changing allocated cloud resources in terms of licensing costs.

3. FURTHER WORKS

In further works we will improve Product Usage Rights identification by proposing normalized standards. SKU itself is not self-sufficient because not normalized and might not describe all details of specific agreement between the

software supplier and software users. It still necessitate to be translated into entitlements and easily transformed to automate compliancy verification. In this aim, we could propose a general model of entitlement's classification handling most of the current and forecasted licensing models.

To go further in our model tests, we will increase its complexity, by implementing more complex licensing rules. We need to strengthen user interfaces and create relevant queries allowing realistic SAM controls and optimization especially considering elastic applications. The qualitative evaluation proposed will be enriched in further works by a quantitative evaluation approach, among others to measure cost of interception of identification metadata and to measure cost of interception of usages.

We believe that an major step in SAM approach will be to move compliancy and optimization control from *a posteriori* (observation of current deployments leading to adapted corrective actions) to *a priori* (before software instantiations). The most advanced usage opportunities provided by the cloud (elasticity, load overflow) add a new dimension to SAM controls. Elasticity consists of being able to switch to other clouds in case of overload; these clouds might have different responsibilities, geographical location or different architectures. It can strongly impact licensing compliance. Overflow consists in punctually requesting all the potentially available resources in a given geographical scope (for example all the resources available in a building, including smartphones, boxes, etc.)… to solve a contention problem. The impact of these evolutions will be even more penalizing in NFV where concerns about bandwidth and response time are crucial and where the potential occurrences are counted by million. There is no work to date on SAM optimization in next-generation cloud environments.

REFERENCED WORKS

[1] Shukla, G., 2017. *Demystifying Accounting for Software Expenses,* s.l.: Ovum.

[2] N.F. Holsing, D. Y., 1999. Software Asset Management: analysis, development and implementation. *Information Resources Management Journal,* Volume 12 Issue 3, pp. 14-26.

[3] M. Ben-Menachem, G. M., 2004. Inventorying information technology systems: supporting the "paradigm of change". *IEEE Software,* Sept.-Oct., 21(0740-7459), pp. 34 - 43.

[4] M. Ben-Menachem, G. M., 2005. IT assets-control by importance and exception: supporting the "paradigm of change". Volume 22, pp. 94 - 102.

[5] L.McCarthy, 2011. Managing Software Assets in a Global Enterprise. *IEEE International Conference on Services Computing,* pp. 560-567.

[6] M.Sharifi, 2009. *A Novel ITSM-Based Implementation Method to Maintain Software Assets in Order to Sustain Organizational Activities.* Athens, IEEE.

[7] M.Ben-Menachem, 2008. Towards management of software as assets: A literature review with additional sources. Information and Software Technology, 50(4), pp.241–258.

[8]P. Klint, C. Verhoef, 2002. Enabling the creation of knowledge about software assets. Data & Knowledge Engineering, 41(2-3), pp.141–158.

[9] M.McCarthy, L.M. Herger 2011. Managing Software Assets in a Global Enterprise. *IEEE International Conference on Services Computing,* pp. 560-567.

[10] A.Manzalini, A. G. W. K., 2015. Softwarization of telecommunications, Special issue : SDN and NFV. *Information Technology,* Issue 10.1515/itit-2015-0025, pp. 321-329.

[11] C.Matsumoto, 2014. *Ciena turns NFV into an Online Shopping Experience.* [En ligne] Available at: https://www.sdxcentral.com/articles/news/ciena-turns-nfv-online-shopping-experience/2014/12/

[12] R. Jones, 2016. *Dynamic licensing for applications and plugin framework for virtual network systems.* US, Patent No. US20160226663 A1.

[13] L.M.Contreras, P. H. D., 2015. Operational, organizational and business challenges for network operators in the context of SDN and NFV. *Computer Network*, 9 December, Volume 92, Part 2, pp. 211-217.

[14] M. Adler, T. R. N., 2014. *Systems and methods for identifying a secure application when conneting to a network.* US, Brevet n° US20140282821 A1.

[15] M.McRoberts, 2013, Software Licensing in the Cloud Age, Solving the Impact of Cloud Computing on Software Licensing Models. The International Journal of Soft Computing and Software Enginnering, Vol 3, No.3

[16]M.Thompson, Practical ITAM – The essential guide for IT Asset Managers – getting started and making difference in the field of IT Asset Management, published by The ITAM Review, July 2017, ISBN 978-1547011216

[17]M.Thompson, 28th Feb 2015, SAM Tool Buyers Guide [On Line] Available at : https://www.itassetmanagement.net/2015/02/28/sam-tool-buyers-guide/. , October 2017.

[18] Aspera Smart track, www.aspera.com/fr/, September, 2016

[19] Snow License Manager, www.snowsoftware.com/fr, September, 2016

[20]Flexera Flexnet Manager, www.flexera.fr/enterprise/products/software-license management/flexnet-manager-engineering-apps/

[21] Spider Brainware, www.brainwaregroup.com/en/solutions/software-asset-management/spider-licence/, September, 2016

[22] Eracent, https://eracent.com/tag/software-asset-management/ September 2016

[23] M.Thomson, 31st July 2013, Review, HP Asset Manager for SAM [online] available at: https://www.itassetmanagement.net/2013/07/31/review-hp-asset-manager-sam/ October, 2017

[24] BMC Remedy, www.bmcsoftware.fr/it-solutions/asset-management.html, September, 2016

[25] GLPI – OCSng, www.glpi-project.org/, September, 2016 & www.ocsinventory-ng.org/fr/, September, 2016

[26] BSA, The Software Alliance, Seizing Opportunity through License Compliance, Global Software Survey 2016, May 2016,

[27] ISO/IEC 19770-2:2015, ISO/IEC 19770-2, Information technology — Software asset management — Part 2: Software identification tag

[28]TagVault Membership list available on www.tagvault.org/about/membership-list/, September, 2016

[29]ISO/IEC 19770-3:2016

[30] IBM Corporation, June 2005, An architectural blueprint for autonomic computing, Autonomic Computing White Paper; Third Edition

[31] S. Kachele, C. Spann, F.J. Hauck, J.Domaschka, 2013, Beyond IaaS and PaaS: an extended Cloud Taxonomy for Computation, Storage and Networking, Utility and Cloud Computing (UCC), 2013 IEEE/ACM 6th International Conference, 9 – 12 Dec. 2013, Dresden, Germany, DOI: 10.1109/UCC.2013.28

[32] www.cloudfoundry.org/, September, 2016

[33] https://github.com/cloudfoundry-incubator/cf-abacus, January 2017

[34] Open Source Mano, An ETSI OSM Community White Paper. Technical Overview, Release Two. - Sophia Antipolis, France: ETSI, 2017.

[35] The Linux Foundation, Harmonizing Open Source and Standards in the Telecom World. The Linux Foundation, 2017.

[36] OPNFV, State of NFV and OPNFV, Study on "What Operators Think of OPNFV". The Linux Foundation, 2016.

[37] Oracle License and Service Agreements [Online]. - 26 August 2010. - 5 May 2017. - http://www.oracle.com/us/corporate/contracts/license-service-agreement/license-service-agreement-070712.html

[38] Microsoft License Review [Online]. - 21 March 2017. - 5 May 2017. - http://www.microsoftlicensereview.com/?p=1159