Software License Optimization and Cloud Computing

Anne-Lucie Vion - Noëlle Baillon
Orange SA
Paris, France
email: annelucie.cosse@orange.com
email: noelle.baillon@orange.com

Fabienne Boyer - Noël De Palma
Univ. Grenoble Alpes, LIG, CNRS
Saint-Martin-d'Hères, France
email: fabienne.boyer@imag.fr
email: noel.depalma@imag.

***Abstract*- In this article, we propose a review of Software Asset Management (SAM) state of the art and existing tools oriented in the Cloud perspective; it seems that Software identification, through its complete virtualized lifecycle, is a major lock in efficient control. In this context, we propose a model and process architecture to cope with this complexity. We underline innovative graph modeling benefits in this contribution. We use a simple, but vivid example to prove the validity of our model.**

Keywords-Software Asset Management; SAM; License optimization; Software uses.

I. INTRODUCTION

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). The current economic climate underlines this particularly burning issue, as each non-compliance situation is heavily penalized in financial aspects.

In this paper, 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. Among others, tracking software becomes more challenging because the 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 (Fig. 1), 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.

	Yesterday	Tomorrow
Software Cycle	Long cycle	Real time
Total costs	Calculable	Hidden and additional costs
Provisioning	Centralized	Built to be decentralized
Expenditure	Organized	Lower financial visibility
Licensing	Complex rules	Complex rules combination
Usage	Understandable	i.e., BYOD, multiplexing
Assets nature	Software	Cloud Services
Virtualization	1 software-1 hardware	Multiple layers: hardware disconnection

Figure 1. Complexity factors brought about by cloud architecture.

The idea that will be developed in this paper is that turning to the Cloud is not changing the object of SAM, but altering how SAM processes should be designed. The contributions are the following. We propose (i) an architecture for SAM in the cloud, (ii) the related SAM management workflow, (iii) some major implementation choices and (iv) a preliminary evaluation.

The remaining of this paper is organized as follows. Section 2 presents a synthesis of the state of the art and our related SAM maturity scale, Section 3 describes our global architecture for managing software, a model for managing installations and usages on PaaS (Platform as a Service) layer and discusses the choice of graph database to support our SAM model. Section 4 presents our first evaluation result, and we conclude in Section 5.

II. STATE OF THE ART

This section discusses the state of the art regarding SAM solutions. We firstly recall the theoretical SAM models and then describe the existing SAM tools. We end this section with a short discussion on the "cloud-ready" dimension of SAM processes.

A. Theoretical SAM Models

One of the first studies leading to SAM considerations, in 1999, was proposed by Holsing and Yen [1] through a software asset probation model and identification of five problem areas, which actuate the need for software management: ethical, legal, technical, managerial and economic.

In 2004, Ben-Menachem and Marliss [2] introduced the "paradigm of change" based on methods, tools and procedures for accurate overall Information Technology (IT) inventory management. Thereby, they underline that investments in the creation and maintenance of a dedicated software inventory is a sine qua non prerequisite to proper long-term SAM.

In 2011, McCarthy and Herger [3] proposed a solution in four points to combine IT, processes and business in SAM perspectives: Discover software assets, mainly consisting in achieving a scan of installed licenses; Reconcile purchased assets, enables performing a procurement inventory; implementing contract management; producing business intelligence reporting "audit readiness" and compliance.

SAM tools are widely used in a lot of computing environments. In 2014, for Gocek, Kania and Malecki [4], these tools refer to software programs that discover and collect information about software instances deployed in monitored environments. As software owners continue to shift toward complex software licensing schemes, SAM tools will continue to play increasingly major roles.

B. SAM Tools

Several studies [5][6][7] show that people around the world, all face the same difficulties to compare existing SAM tools. This is mainly due to the exuberant marketing made by publishers about features that appear similar between existing tools and the lack of a model to classify them. The scope is absolutely not defined between traditional architecture and the cloud environment, as if the way to manage software assets in both environments was similar. We can add that multiplication of tools

 ISBN: 978-1-61208-529-6

is also due to multiplication of actions to manage: i.e., management tools often perform discovery activities and inventory. However, they rarely gather sufficient details on software inventory to allow making informed decisions about their elimination or even just to compare to usage rights acquired by contracts.

We have developed the following SAM maturity scale, illustrated in Fig. 2, to compare the existing SAM tools regarding the features they provide. In Fig.2, four levels are defined on a **vertical axis** about SAM maturity.

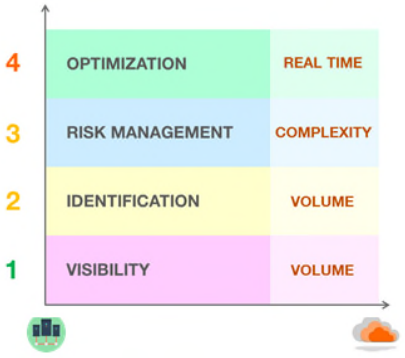

Figure 2. SAM processes maturity scale

(1) *VISIBILITY:* this level precisely identifies resources. In other words, SAM tools providing this level of feature allow (i) recognizing each device with its physical features, (ii) identifying the virtual machines and the resources allocated to them, and (iii) discovering any software installed on any physical or virtual devices.

(2) *IDENTIFICATION*: this level consists in translating any resource in its associated assets. In other words, it translates 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 belongs to a software suite or that it is an option whose use is conditioned by the use of the basic product.

(3) The third level, *RISK MANAGEMENT* consists in reconciling data provided by the two previous levels: VISIBILITY and IDENTIFICATION. In other words, the goal is to compare product usage rights with real uses. Mainly, the aim is to prevent two different risks: the first one is a legal one, counterfeiting: using software without license or with wrong way of licensing (nowadays, more often due to the complexity of licensing models). The second is a financial risk, over-deployment: not using licensed software, or the license covers more usage rights than needed.

(4) *OPTIMIZATION*: through the accurate view of installations, usages and assets provided by the previous levels, it becomes possible to identify ways to improve license spends, and in fine to automate this optimization process in a real time manner.

Fig. 3 illustrates the current state regarding existing SAM tools. We can notice that the four levels previously introduced do not have the same maturity. A lot of tools are really efficient at the VISIBILITY level, in terms of discovery of resources on equipped resources. Among others, we can cite BladeLogic [8], Open Computer Software Inventory New Generation [9] (OCS), System Center Configuration Manager [10] (SCCM).

Figure 3. Main functionalities and limitations of most popular SAM tools

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. At this IDENTIFICATION level, we find tools like GLPI (Gestion Libre Parc Informatique) [11] that manage resources discovered in the first step, but are not able to truly identify product usage rights. Contrary, tools proposed by Aspera [12], Snow [13] or editors' own solutions are able to manage product user rights (PUR) and, for some of them, able to identify some risks of over/under deployments (Snow, Spider [14], Aspera). However, these tools are expensive, especially database updates, and do not offer complete lifecycle tracking.

It is important to mention that software identification mainly relies on tags (i.e., SoftWare Identification Tag (SWID tag) [15]) that record unique information about an installed software application, including its name, edition, version, whether it's part of a bundle and more. The structure of SWID tags is specified in the international standard ISO/IEC 19770-2:2015 [16], which defines an XML (eXtensible Markup Language) data structure aiming to the precise identification of software, regardless of the platform and the device on which it is installed.

Finally, regarding the last level, in the current situation, despite the numerous risk management tools, the treatments are still approximate and optimization difficult to automate.

C. Synthesis

One of the business benefits of cloud computing is its agility and speed-to-market. Services are provisioned, configured, released in a matter of minutes. Thus, while traditional SAM processes assume long lifecycles (usually, we can consider 5 – 8 years for a software, it leads to long cycles of contract, discovery, inventory and reconciliation), cloud accelerates these processes up to real-time requirements.

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

Moreover, we underline that if SaaS (Software as a Service) seems to reduce or even delete infringement risks (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 appropriate controls to ensure compliance with all requirements and limitations (geographical scope, Restriction on shared accounts, on non-employees/providers, partners, etc., time, transactions volume). 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 Business Software Alliance (BSA), 2014 [17] cloud services are often considered as operational expenses and not as capital expenditures. It leads to several problems: (1) less involvement in the contracting phase, (2) loss of control of operational dependencies, (3) loss of known limits to final costs, (4) lack of financial visibility, and (5) increased license compliance risks.

III. PROPOSITION OF A SAM MODEL FOR THE CLOUD

A. SAM 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 (or SAM check-points). All possible information related with the use of software should be captured and stored in order to implement all the required usage controls.

Through the check-points, the SAM processes analyze the current situation in real-time and confront the use of services with the license stock. SAM processes also take potential optimization decisions, creating a control loop.

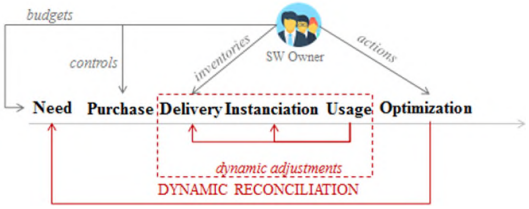

Figure 4. SAM retroaction loop

In its basic form, the software lifecycle that we consider is composed of 5 + 1 steps as shown in Fig. 4 and Fig. 5 – Fig.9; some steps can be played several times:

1) *Need's Expression*: the consumer justifies his need and choice of software.

2) *Purchasing*: this step encompasses sourcing processes, negotiation, contract, billing etc. At this stage, we get a Stock Keeping Unit (SKU) identifying the purchased software and its own [product] usage rights (PUR) created by the manufacturer and acquired during purchasing processes.

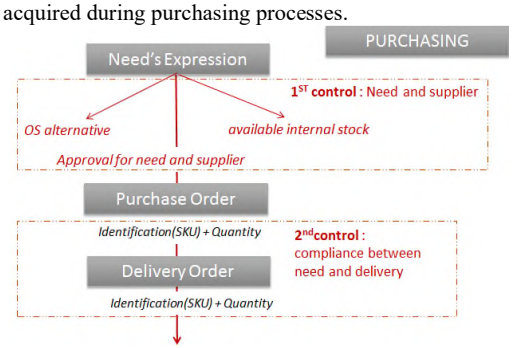

Figure 5. SW lifecycle and SAM controls - Purchasing

3) *Delivery*: this step corresponds to the software receipt via downloading platforms, preparation for installation on user platform, entry into a software catalogue. Through this step, we get a SWIG Tag containing the software's SKU created by the manufacturer and extendable with client-specific information. SWID tag will be the default software identifier.

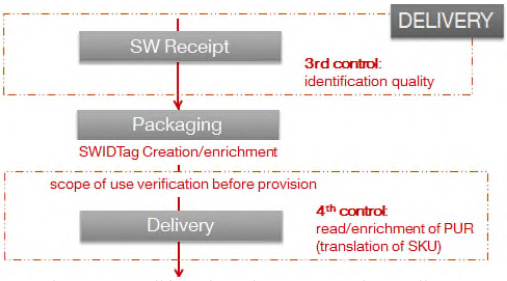

Figure 6. SW lifecycle and SAM controls – Delivery

4) *Instantiation*: The software is installed in an environment (for instance, a given Cloud), in other words, the software is able to be used.

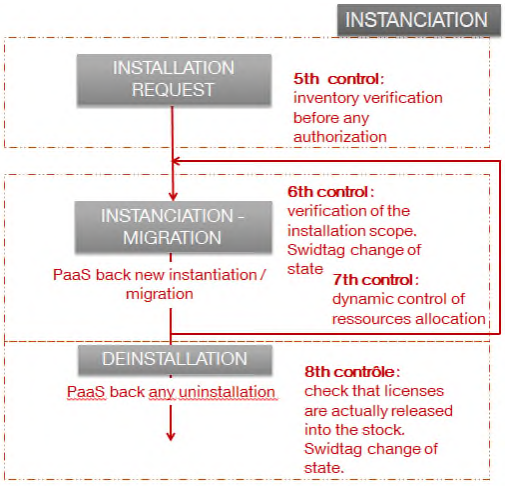

Figure 7. SW lifecycle and SAM controls - Instantiation

5) *Usage*: a user consumes a service/software. Here, we have to identify the cases where multiple users consume the same service simultaneously and translate this in terms of use (multiplexing, multidevice , etc.)

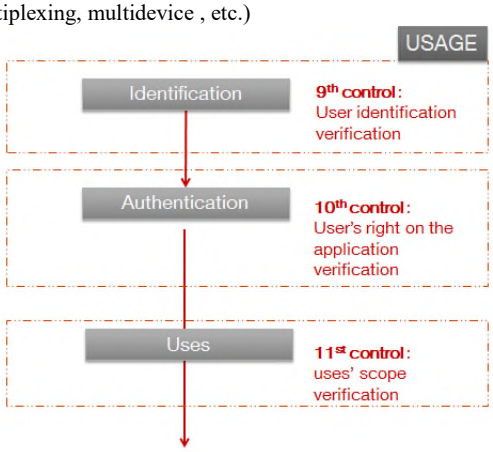

Figure 8. SW lifecycle and SAM controls - Usage

6) *Optimization:* this 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.

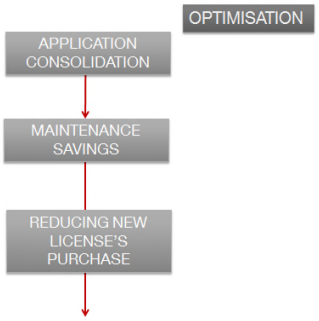

Figure 9. SW lifecycle and SAM controls - Optimization

 ISBN: 978-1-61208-529-6

B. Instance and usage capture

1) Focus on Instantiation

To implement SAM check-points over the instantiation step, we need to make assumptions on the targeted cloud environments, especially in terms of the 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 [18] PaaS. 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 further works, we will extend to Infrastructure as a Service (IaaS) layers.

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 SWID Tag enriched by all relevant contractual information during delivery step)

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.

2) Focus on usage

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 detail (Fig. 10), 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 certificates, the user can start to consume application

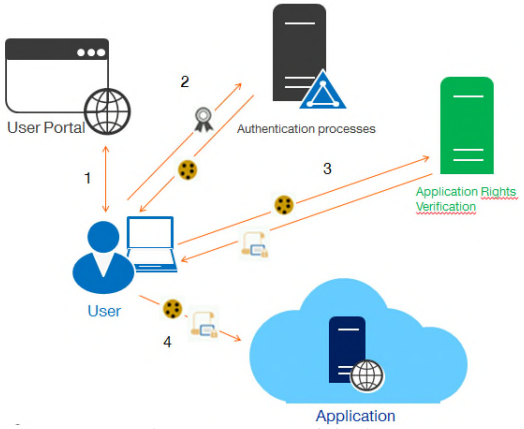

Figure 10. Use case of cloud app access

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.

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 [19] provides usage metering and aggregation for Cloud Foundry services. 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.

We should recall that SAM's purpose is to confront contractual provision (PUR) with observed usages. Since we assume usage capture, we should focus on this aim to direct our implementation choices. Indeed, how we store the collected information (instantiation + usage) influences comparison and optimization operation's efficiency and relevance.

C. Feeding the database

Each step previously described feeds a software database (SWDB). Following software lifecycle, we can represent every data injection summarized in Fig. 11:

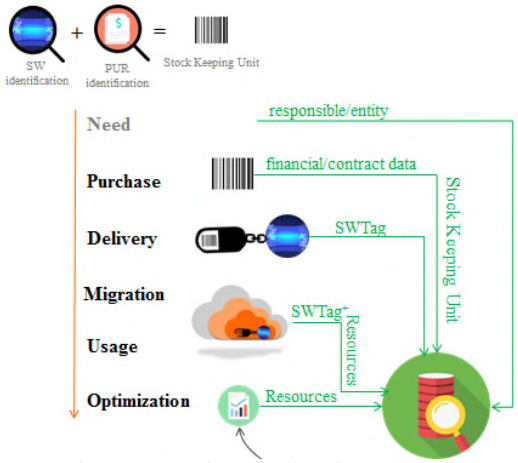

Figure 11. Asynchrone feeding of SAM database

To implement this database, we adopted a graph-oriented database. The lack of flexibity is the biggest weakness of relational databases when the data structure may vary like for SAM topic. Their scheme cannot support the dynamic real time, and uncertain nature of data, new technologies and platforms. Graph data models are centered on relationships. Just by connecting nodes and relationships, it can generate sophisticated models that fit closer to our problem (cohesive picture between contracts, usages and installations) when relational databases require us to infer connections between entities using special properties such as foreign keys, or out-of-band processing like map-reduce disconnecting the evolving schema and the actual data model.

Each node in the graph database model contains a list of relationship-records. These relationship records are organized by type and direction and may hold additional attributes (Fig. 12). Whenever one runs 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.

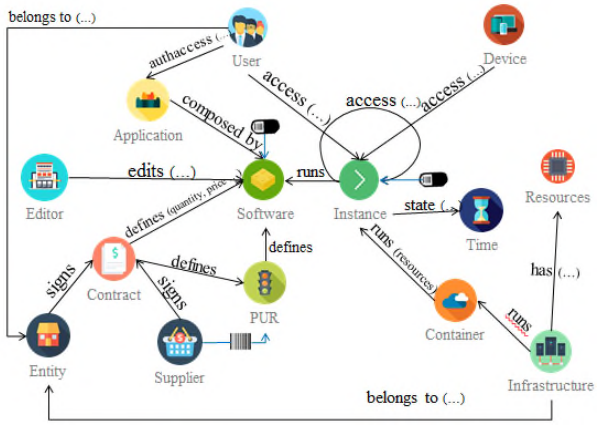

Figure 12. Graph modelization of SAM in Cloud

Graph representation makes the comparison between all software dimensions easier: looking at this model, software is logically linked to contracts, instances and user. The global picture seems to be cohesive and we can identify Software lifecycle, likewise tag's cycle. It can be read like: "Entity signs (a) Contract, which defines Software run by (a/n*) Instance(s) etc."

IV. EVALUATION

In our preliminary evaluation, we focused on the purchase-delivery-instantiation-usage phases of our SAM model, omitting the optimization phase that remains a future work. Our objective was to validate the fact that our graph model can be managed through a capture of PaaS usages (Cloud Foundry/Abacus in our experience). To achieve this experience, we considered a well-known software: an Oracle database.

We choose the Oracle Database Enterprise Edition (Oracle DB EE) example for several reasons: (1) It is a vivid example for the SAM community; one of the most often mentioned for the complexity of its license management. (2) Oracle DB licenses can be defined by several types of metrics, oriented on material (i.e., CPU) or user (i.e., Named User Plus). (3) 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.

In this theoretical evaluation, we will follow the Software lifecycle proposed in Fig.4 and Fig.5-Fig.9 and refer to the Fig.2 SAM processes maturity scale: visibility, identification, risk management and optimization.

1) *Purchasing*

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

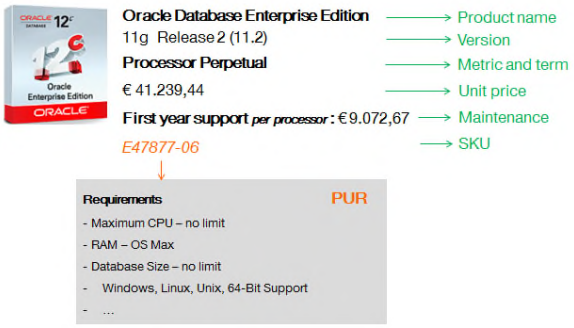

Figure 13. Example of Oracle's offer

Fig. 13 can be an extracted from "License Store's" catalogue proposing the product we need and are planning to buy.

Few elements (above) are necessary to identify precisely this offer and determine the level of grants (PUR) given by this way of licensing. These elements have to be collected in the purchase order and reconciliated with data from the delivery order. In the graph: 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 'Retailer' and 'Editor' to identify a node 'License Store' and 'Oracle':

```
CREATE (m:Software { name : 'Oracle Database', version : '11g Release 2 (11.2)', sku:'E47877-06', category: 'Database'})
CREATE (z:Supplier {name : 'License Store'})
CREATE (n:Editor { name : 'Oracle' })
```

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

```
CREATE (o:PUR {metric: 'processor', term : 'perpetual'})
CREATE (p:PUR {name: 'requirement', maximumCPU : 'no limit', RAM : 'OS max', DatabaseSize : 'no limit'})
CREATE (q:PUR {name: 'OperatingSystem', windows : 'yes', unix : 'yes', linux : 'yes'})
```

Then, we create relations between nodes:

(1) Between an editor and product (EDITS): 'Oracle' edits 'Oracle Database'.

(2) Between a product and PUR (DEFINES): 'Oracle DB' is licensed under processor metric/ or can run on windows/Unix/Linux/etc.

(3) 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 the 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.

2) *Provisionning*

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, let us create a label Application and a node 'HumanRessources' which we'll include in our Database.

The relations 'CONTAINS' is enriched by properties like a project's id or application's project manager:

```
CREATE (n:Application {name :'HumanRessources', responsible :'Tom'})
MATCH (a:Software),(b:Application)
WHERE a.name = 'Oracle Database' AND b.name = 'HumanRessources'
CREATE (b)-[r:CONTAINS  {id_project : '1234R'}]->(a)
```

3) *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 resources (Virtual Machine (VM), networking, storage), database instantiation, Subscription to shared services, application deployment, installation, configuration, application monitoring, application log collection and interaction with app-ops (inventory/CMDB, monitoring/alerting)

CF allows identifying allocated resources. Our experience is here restricted to PaaS layer, it would be necessary to reach underlying infrastructure (i.e., VMware, OpenStack etc.) to obtain IaaS resources. All this chain allows to keep and know information about the allocated ressources in each stage.

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 contains metadata like SWIDTag enclosed during the provisioning. A key part is now to create links between instance and product which we bought. Everything is based on the use of SKU number. The instance knows and updates all SWIDtags of its components (i.e., Fig 14). This allows to create the link between the product in catalogue and the installed product.

Balise	Description
entitlemet_required_indicator	true if activated/serialized false if evaluation/not licensed
product_title	Oracle Database EE
product_version • name • numeric	 • chaine version • numeric
software creator • name • regid	 Oracle regid.1977.oracle
software licensor • name • regid	 Oracle regid.1977.oracle
software_id	E47877-06
tag_creator • name • regid	 Oracle regid.1977.oracle
• license_linkage • activation_status	evaluation/serialisez/actived/abonnement/no license

Figure 14. SWID Tag example for Oracle DB EE

```
MATCH (i:Instance),(a:Software)
WHERE i.software_id= a.sku
CREATE (i)-[c:INSTANCES]->(a)
```

4) *Uses*

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.

The link ACCESS/AUTHACCESS has properties that characterize the use (scope, duration, consumed resources, etc.) captured by CF and accumulated/aggregated by Abacus. The capture fulfills the step 2 of the SAM processes maturity scale.

```
MATCH (s:Software {name:'Oracle Database'})<-[:INSTANCES]-
(i:Instance)
RETURN s, i
```

"Show me all 'INSTANCES' relation(s) to 'Oracle Database'" will provide all instances related to Software. As we can identify the Product Usage Rights (via the SWIDTag/SKU) by a direct link between Software/PUR and Software/Instances, we can fulfill first part of the step 3 (SAM processes maturity scale): the risk management (here: counterfeit risk).

```
MATCH (i:Instance {name:'HumanRessources'})<-[:ACCESS]-(user)
RETURN i, user
```

"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).

5) *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 us study quickly three of the most basic, but also the most important:

- *What I bought ?*

```
MATCH (e:Entity)-[g:SIGNS]-(c:Contract)-[r:DEFINES]->(s:Software)
MATCH (p:PUR)-[h:DEFINES]->(s) WHERE p.name='Metric'
RETURN e.name AS Entity, s.name AS Software, s.SKU AS SKU,
sum(r.quantity) AS Quantity, p.metric AS Metric,
c.date AS Date ORDER BY e.name, s.name, c.date
```

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

- *What I instanciated ?*

This query returns a table: the number of instances per software ordered by metric with collection of application containing this software.

```
MATCH (v:VM)-[t:RUNS]->(i:Instance)-[r:RUNS]->(a:Application)-
[c:COMPOSEd_BY]->(s:Software)
RETURN s.category AS Category,s.name AS Software, s.SKU AS SKU,
count(t) AS InstanceNumber, collect(distinct(a.name)) AS Application
```

- *Am I compliant ?*

This last query consists in a verification of the 'Processor' metric (typical for Oracle). Basically, we have to multiply the number of cores per processor of the physical machine hosting the DB by the number of processors and by a coefficient given by Oracle for each processor. It returns a table of licenses ordered by the software, and he number of bought/instanciated (according to Oracle licensing rules).

```
MATCH (s:Software)  WITH s
MATCH (m:Machine)-[*]->(a:Application)-[co:COMPOSEd_BY]->(s)
MATCH (r:Resources)<-[h:HAS]-(m:Machine)
WITH s,((toFloat(r.core))*(toFloat(r.corefactor))*(toFloat(h.quantity))) AS
nbL, m,r
RETURN s.name as Software,s.SKU as SKU,
SUM(nbL) AS ProcessorLicenses
```

6) SAM Optimization

Optimization consists, first, in automating the rise of alerts. When a countfeiting situation is detected (piracy, but mainly editor's metric misunderstanding) or when the 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 of operating simulations: usage/instantiation captures, may reveal some possibility to renegociate a contract in a more favorable (financial) way: i.e., to change the Oracle DB negociated metric (currentlypProcessor) into another metric (i.e., Access), more appropriated to observed uses, or to project a future software/license uses based on current observed situation. This will be developped in further works.

V. CONCLUSION

Software is not like other IT assets that can be considered as just software installation or in its intangible dimension provided by the license that defines the scope of use. Both dimensions of the software should always be considered in managing this asset. Virtualization and Cloud technologies add a new degree of complexity in the first dimension (material) when 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 a disconnection between the software material and intangible dimensions.

We point out in the article the problem of software identification through its complete lifecycle and proposed a reference model or architecture for SAM to cope this complexity. This reference model also help getting a clear understanding on how SAM can be applied in the cloud computing domain. Using the Oracle DB example, we assess that our model works on simple, but vivid SAM cases, and that choice of a graph model is relevant.

Next steps will be (1) to increase complexity of the model, by implementing more complex licensing rules (2) to show more complex interface and queries allowing realistic SAM controls and optimization; (3) to measure cost of interception of SW tags; (4) to measure cost of interception of usages. Regarding step (2), considering elastic applications will be a major step.

REFERENCES

[1] F. N. Holsing, and D. Yen, "Software Asset Management: Analysis, Development and Implementation", *Information Resources Management Journal (IRMJ) 12(3)*, pp.13, 1999

[2] M. Ben-Menachem, and G.S. Marliss, "Inventory Information Technology System: Supporting the « Paradigm of Change »", *IEEE Software*, pp.34-43, 2004

[3] M. McCarthy, and L.M. Herger, "Managing Software Assets in a Global Enterprise", *IEEE International Conference on Services Computing*, (pp. pp.560–567), 2011

[4] P. Gocek, P. Kania, B.Malecki, M. Paluch, and T. Stopa, "Obtaining software asset insight by analyzing collected metrics using analytic services", *US9424403 B2*, Available from https://www.google.com/patents/US9424403, 2014

[5] J. Disbrow*,"*Software vedor auditing trends : What to watch for and how to respond", *Gartner (DOI G00230816),* 2012

[6] J-D. Lovelock, Worldwide IT Spending Forecast", *Gartner (DOI: G00323753),* 2016

[7] C. Rudd, ITIL V3 Guide to Software Asset Management. Broché, 2013

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

[9] www.ocsinventory-ng.org/fr/, September, 2016

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

[11]www.glpi-project.org/, September, 2016

[12]www.aspera.com/fr/, September, 2016

[13]www.snowsoftware.com/fr, September, 2016

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

[15] www.tagvault.org, September, 2016

[16] ISO/IEC 19770-2:2015

[17]A.Hughes, Seizing Opportunity Through License Compliance. BSA, Software Alliance, 2016

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

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