Automatic Configuration Management: Autodiscovery of Configuration Items and Automatic Configuration Verification

Nadine Perera[1]
German Aerospace Center (DLR), 82234 Wessling-Oberpfaffenhofen, Germany

Performing Configuration Management (CM) on systems such as an entire space mission ground system is essential. CM verifies that a system performs as intended, and is identified and documented in sufficient detail to support its projected life cycle. However, manual Configuration Management and Change Management on all the hardware, software, and documentation items comprising a complete ground system is time consuming and error-prone. An automated configuration discovery approach, where hardware systems are scanned for software and the results reported into a CM database, limits human error and saves a lot of time by reducing the need for manual configuration management. This paper describes the approach taken at GSOC during the preparation of two geostationary communication satellite missions and low earth orbit satellite missions to automate the task of hardware and software CM. This makes CM more efficient and transparent, and enables the Configuration Manager to track the configuration in the CM database during the full cycle, from configuration deployment to configuration verification.

Nomenclature

CI	=	Configuration Item
CIDL	=	Configuration Item Data List
CIL	=	Critical Item Data List
CM	=	Configuration Management
CMS	=	Configuration Management System
ECR	=	Engineering Change Request
GITS	=	GSOC Issue Tracking System
GSOC	=	German Space Operations Center
IP	=	Internet Protocol
LAN	=	Local Area Network
OCS	=	Open Computer and Software (Inventory Next Generation)
OPS	=	Operations
RAM	=	Random Access Memory
RPM	=	Red Hat Package Manager
SCOS	=	Satellite Control and Operation System
SLES	=	SUSE Linux Enterprise Server

I. Introduction

Configuration management is a process closely linked to *change management*, which is also called *configuration control*. Any system that needs to be controlled closely and run with good reliability, maintainability and performance benefits greatly from configuration management, i.e., the management of system information and system changes. Configuration management can extend life, reduce cost, reduce risk, and even correct defects. It should be applied over the life cycle of a system in order to provide visibility and control of its performance as well

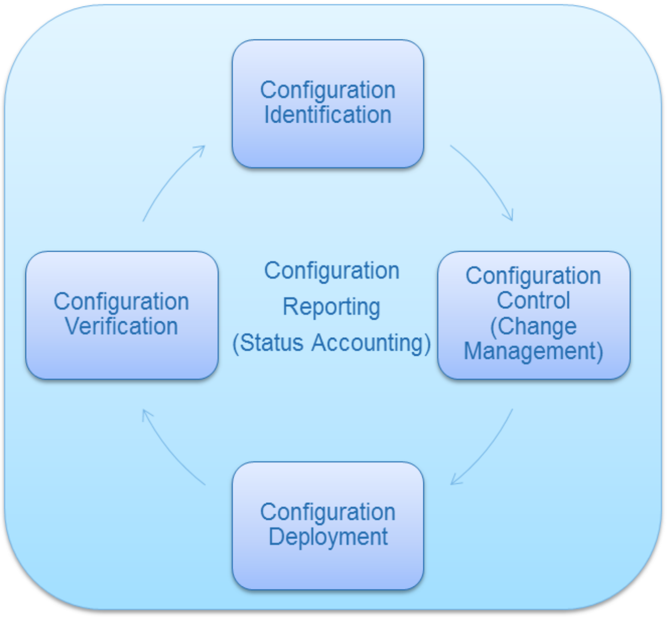
[1] *Configuration Manager, Quality, Configuration and Security Management, GSOC, Münchner Straße 20, 82234 Wessling-Oberpfaffenhofen, Germany, Nadine.Perera@dlr.de*

as its functional and physical attributes. Configuration management verifies that a system performs as intended, and is identified and documented in sufficient detail to support its projected life cycle. In order to effectively control system change, it is important to model the functional relationships between system parts, components, and subsystems. This facilitates the systematical consideration of changes with the goal of minimizing resulting errors. Change control means that system changes are proposed, evaluated, implemented and verified using a standardized and systematic approach, ensuring consistency and minimizing the risk of changing parts of the system inadvertently, because the system was not considered in its entirety.
In the case of non-conformances, configuration management and change control support fault analysis, as errors occur in a specific system configuration or may be introduced by changes (or the lack thereof). Configuration baselines are recorded for system validations and system tests, such that one knows how the system undergoing the test was exactly configured at that point in time. For the possibility of disaster recovery, configuration control can help to re-establish the last known system configuration, provided that the configuration data survive the disaster. The automatic configuration approach described in this paper has been applied in a pilot project to the geostationary EDRS[2] and HAG1 missions, the low Earth orbit TanDEM-X mission, and the Multi-Mission project at GSOC.

Configuration management is a vital concept both in the *system* engineering and in the *software* engineering discipline. When software engineering is combined with system engineering, e.g., because a control center is building a ground system that has software components, which are developed in-house using a software engineering process, it is important to clarify the context. Different tools and processes are applied for configuration and change management of *the ground system*, consisting of hardware, software, and documentation, and *a software system*, consisting of files of source code and possibly libraries and artifacts, and constituting one software configuration item of the whole ground system. We discuss configuration management in the context of *systems engineering*.

A. Configuration Management Activities

Configuration management is often depicted[1] as a sequential process of the activities *Configuration Identification*, *Configuration Control and Change Management*, *Configuration Status Accounting*, *Configuration Reporting*, *Configuration Verification and Audit*. In practice, configuration management is a cyclic process, similar to the Plan-Do-Check-Act continuous improvement cycle known from quality management[2], cf. Figure 1: *Configuration Identification*, *Configuration Control* and *Change Management*, *Configuration Deployment*, and *Configuration Verification* are activities that happen sequentially, while *Configuration Status Accounting and Reporting* is undertaken during all stages of the cyclic process. The process starts again with *Configuration Identification* based on deviations encountered at the *Configuration Verification* stage, or with configuration items that are added to or removed from a mission during its lifespan.


Configuration Identification
Configuration Verification
Configuration Reporting (Status Accounting)
Configuration Control (Change Management)
Configuration Deployment
Figure 1: Configuration Management Activity Cycle

Configuration Identification is an activity usually associated with the creation of the *product tree*, the list of *Configuration Items*, i.e., hardware, software and documentation that will make up a project's infrastructure, at GSOC usually a ground segment for spacecraft operations. The project manager has to plan, name, and procure the assets that are intended to operate the mission. Sometimes, the configuration items are defined based on a Critical Item List (CIL), since all potentially critical items need to be defined in that list as well.

[2] https://en.wikipedia.org/wiki/PDCA

Configuration Control and Change Management are terms used synonymously. Controlling the configuration means that changes to the configuration are managed and documented. At GSOC, we use GITS (the GSOC Issue Tracking System) to write and process Engineering Change Requests (ECRs) for change management on the ground system configuration items. An ECR in GITS specifies at the very least the "system affected", i.e., the mission subsystem that the change affects, more detailed information may be entered in the field "system affected version". This information allows the responsible configuration manager to link the affected Configuration Item in the CMS to the change report, such that all changes to a CI, and all CIs affected by a change, are documented.

Configuration Deployment is an activity that is usually not considered as part of the configuration management process, or just as a part of change management. The approach undertaken at GSOC relies to a certain extent on automatic configuration deployment; therefore, we chose to list this activity specifically. Typically, there are an OPS (operational), a SIM (simulation), and a TEST chain of hardware CIs with associated software CIs in an operational setting, and quite often, a standard configuration for hosts is required. Automatic configuration deployment tools are tailored to the need of rolling out operating systems, software package combinations and configurations to multiple hosts.

Configuration Verification is the activity of verifying that the configuration depicted in the CMS concurs with reality, i.e., what is actually installed and configured on the configuration items. It has to be verified that the correct versions of the software CIs are installed on the correct hardware CIs, and that *only* the correct versions of the software CIs are installed there (and not anywhere else).

Configuration Status Accounting and Reporting is the activity of generating reports such as the Configuration Item Data List (CIDL), and creating configuration baselines, also called snapshots, which list the system configuration at a specific point in time. This creates the option to compare the configuration of two points in time by comparing the two snapshots, generating a list of configuration items that have been added or removed in the time between the two dates. The parameters of a configuration baseline can be connected into a URL, such that the baseline within the CMS can be reached directly by clicking the link and logging into the CMS. This URL is helpful in order to store the system configuration with test reports, as described in Ref. 3.
In Figure 1, Configuration Reporting and Status Accounting is shown as an activity that is ongoing in the background and not a step in the cycle of configuration identification, control, deployment and verification, since the status of configuration items can always be reported, even while all the configuration items are still only in planning. In fact, "In Planning" is in fact the first lifecycle status that a Configuration Item can have, cf. Section B5, Lifecycle States.

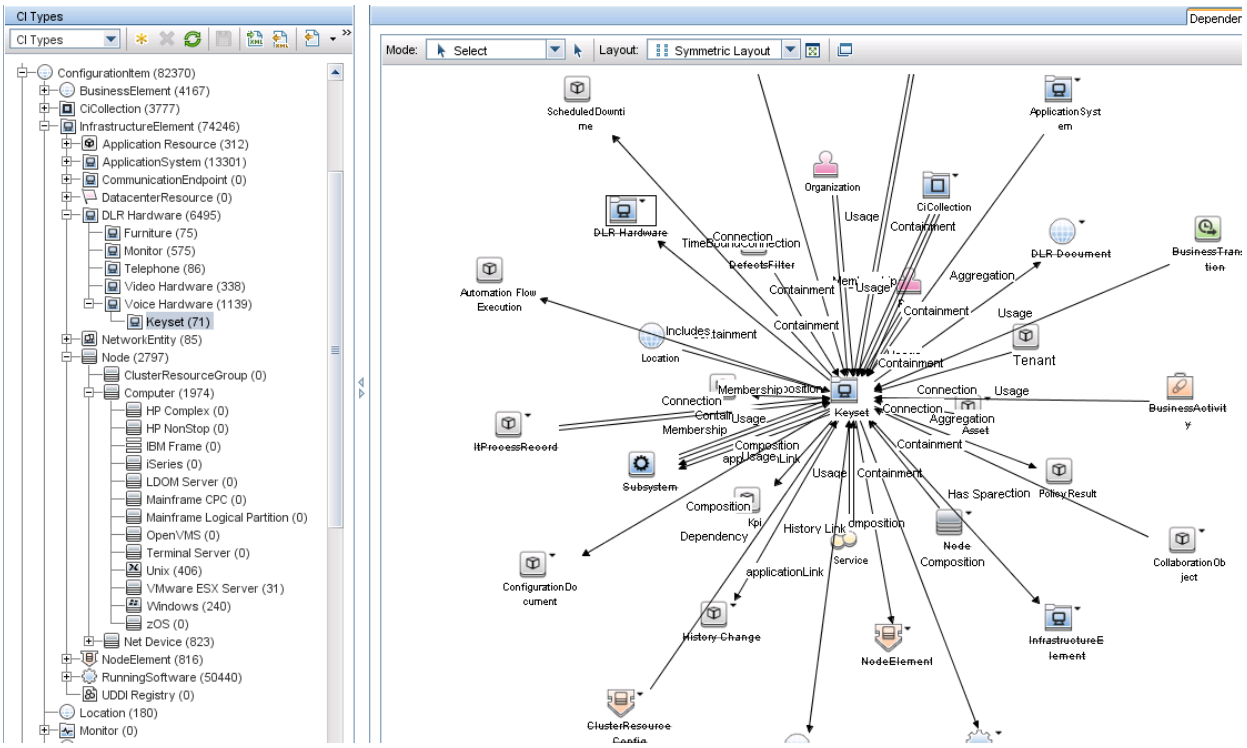
B. Configuration Management System HP uCMDB

The Configuration Management System (CMS) in use at GSOC is the Hewlett Packard Universal Configuration Management Database; HP uCMDB[3]. With the HP uCMDB, CM data can be edited manually via a GUI, manipulated in batch imports and exports via CVS and XML files of a certain format, and configuration baselines can be compared in a graphical user interface. A "Related CI View" is available in the detailed view for each CI, which shows the CIs with a relation to the currently viewed CI.

1. Entity Relationship Data Model

The entity and relationship model[4] is a very popular data model and implemented by our configuration management system. The entities of configuration management are called Configuration Items (CIs) and are commonly of the type hardware, software, or documentation. However, quite a few other types exist, e.g., location. The entities or CIs are structured in a tree of configuration types that allows different shades of specification, for instance, all CIs are of the type *Configuration Item*, some of them are of the type *Hardware*, which is a more specific type, and of the hardware CIs, some are *hosts*, while others are *routers* or *switches*, and yet others may be a specific *voice system hardware*. Figure 2 shows part of the CI Type hierarchy, and some of the many relationship types modeled in the HP uCMDB.

[3] http://www8.hp.com/de/de/software-solutions/software.html?compURI=1172882#.UxcKzIXy2HM
[4] https://en.wikipedia.org/wiki/Entity%E2%80%93relationship_model



```
CI Types
CI Types
ConfigurationItem (82370)
  BusinessElement (4167)
  CiCollection (3777)
  InfrastructureElement (74246)
    Application Resource (312)
    ApplicationSystem (13301)
    CommunicationEndpoint (0)
    DatacenterResource (0)
    DLR Hardware (6495)
      Furniture (75)
      Monitor (575)
      Telephone (86)
      Video Hardware (338)
      Voice Hardware (1139)
        Keyset (71)
    NetworkEntity (85)
    Node (2797)
      ClusterResourceGroup (0)
      Computer (1974)
        HP Complex (0)
        HP NonStop (0)
        IBM Frame (0)
        iSeries (0)
        LDOM Server (0)
        Mainframe CPC (0)
        Mainframe Logical Partition (0)
        OpenVMS (0)
        Terminal Server (0)
        Unix (406)
        VMware ESX Server (31)
        Windows (240)
        zOS (0)
      Net Device (823)
    NodeElement (816)
    RunningSoftware (50440)
    UDDI Registry (0)
  Location (180)
  Monitor (0)
```

Mode: Select
Layout: Symmetric Layout

Figure 2: CI Type Hierarchy: *the left side shows the hierarchy of CI Types, the right side details the relationships of the CI Type Keyset to other Configuration Items.*

2. *CI Attributes and Relationships*

All CIs have a list of attributes that can be defined on different levels of the CI type tree, e.g., all configuration items have a *name* and an *ID* attribute, but only documentation CIs have an *author* attribute, while hardware and software CIs can have a *manufacturer* and a *serial number* attribute. Software and document CIs have a *version* attribute and a *release date*, or *install date* in case of the software, attribute. Attributes are filled with values, which may be of different types, such as text of a certain length, numbers, or enumeration values.
The CIs can be in relationships with one another, the most interesting relationship in this context is the software *isInstalledOn* hardware relationship. All relationships are bidirectional, which means that if a software CI has an *isInstalledOn* relationship to a hardware CI, the hardware has the reverse *hasInstalled* relationship to the software CI. Rather than model the location of physical CIs, i.e., hardware CIs, with an attribute, the locations are modeled as entities, such that entering the location information for a hardware CI by creating a relationship between hardware and location will also provide the information to the room which hardware CIs are located there. The benefit of using entities and relationships instead of storing the information into entity attributes can be seen at the example of software to hardware relationships: one single hardware CI can easily have over one thousand software CIs installed, of which many can be installed on hundreds of other hardware CIs. Documenting those lists of related configuration items in an attribute field would be difficult to impossible, and not allow for the bidirectional information flow that relationships provide. In regard to virtualization, host CIs may be physical hardware or virtual machines clustered on ESXi Servers. Figure 2 shows all the relationships that a voice system keyset can have with all the other CI Types. The *isInstalledOn* relationship (not shown in Figure 2) is considered the most relevant for configuration automation.

3. *Mission vs. Multi-Mission CIs*

GSOC has a multi-mission control center strategy, meaning that some resources are shared among different satellite missions, or even between manned missions and satellite missions. The shared resources in the CM context are called multi-mission configuration items and may be any of the CI types, e.g., rooms, hardware, or software. Everything that is not mission-specific is by definition multi-mission. Multi-mission CIs that are part of a particular mission configuration may be linked under all the individual missions which make use of them. Linked CIs are part of the CIDL of the mission(s). The definition of a software CI as part of the CIDL of a mission is that there is at least one piece of hardware that has this software CI (in the version given in its version attribute) installed at that point in time.

4. Modelling CI Versions as Single or as Separate CIs
Software or documentation CIs both have a *version* attribute, consisting of a text field into which text can be entered, typically (but not enforced) of the form x.y.z, with x,y, and z being natural numbers including 0. For a documentation CI, it makes sense to change the version number with any update of the document, i.e., from 1.0.0 to 2.0.0. The document CI is one instance, and the old version 1.0.0 will lose its validity as soon as version 2.0.0 is approved. However, for software CIs, the case is not that clear. If the software CI exists in one instance only, and the change from one version to the next one affects all the hardware where this software CI is installed, it is valid to just update the software CI version number. However, if several instances of the software CI are installed on some hardware items at the same time, and the update only affects some of the instances, we clearly need an additional software CI to model the other, coexisting versions, as one single software CI instance can only have one version at a time. In a control center setting, where there are typically an OPS, a SIM, and a TEST chain of redundant hardware components, it is quite often the case that a new software version will be rolled out onto the TEST chain first, while the other chains remain with the older software version until the new features have been properly tested. In this case, more than one instances of the software CI are in use. This should be reflected in the CMS.
The consequence of modeling a series of software CI versions as separate software CIs is that the software CI history will not show the version changes at a glance, as is the case for single instances of CIs whose version attribute is changed with every update. However, via the Related CI View, one can immediately see which software CI versions are installed on which hosts, and in the history of the host, the history of software CI versions including the installation date remains visible.

5. Lifecycle status of CIs
One important attribute that all software and hardware CIs in our data model possess is the lifecycle state. It is an enumeration attribute, i.e., one value can be selected from a list of possible states:

1. *In Planning:* there is no physical CI yet, only the idea of a hardware or software component
2. *In Purchase:* physical machines or operating system licenses for virtual machines, or COTS (Commercial Off The Shelf) software is being ordered
3. *On Stock:* the CI is on site, but not part of the active configuration, e.g., hardware spare parts or undeployed software CIs
4. *Under Development:* a hardware is being assembled or a software CI is being developed
5. *In Test:* the CI is installed in a TEST chain, but not part of the operative configuration
6. *Active:* the CI is part of the operative configuration
7. *In Repair/Maintenance:* a hardware CI is being repaired or a software CI has been deinstalled to be overhauled
8. *Phased Out:* a hardware CI is not part of the active configuration any more, but still on the premises, or a software CI has been deinstalled, both are waiting to be sold or scrapped.
9. *Removed:* the CI is removed from the premises.

CIs that are set to status “9 – Removed” are not shown in the default CMS view, but there is a view including the removed items that can be switched on if necessary. In Planning is the default status for manually created CIs. The logical status transition diagram for hardware and software CIs is shown in Figure 3.
Since one CI can only ever have one state at a time, a software configuration item is active as long as there is at least one hardware that has an isInstalledOn relationship to the software CI. At the moment where the last isInstalledOn relationship is severed, because the software CI is deleted from the last hardware CI, the software CI ceases to be part of the system configuration.

6. Current CI numbers
Currently, there are about 32,000 CIs modeled in the CMS. Of these, 2,974 are host computers, of which 977 are in lifecycle state active today. There are 13,255 software CIs in the CMS.
Within the last year, the Autodiscovery has created 10,948 of the total 13,255 software CIs. Only 1,646 of the 10,948 software CIs are in lifecycle state active today, with 49,576 automatically added *isInstalledOn* relationships.
By comparison, the manually modelled CIs from the last 5-8 years consist of 2,307 of the total 13,255 software CIs. Of the 2,307 manually added software CIs, 2,140 are in lifecycle state active today, with 749 manually added *isInstalledOn* relationships.

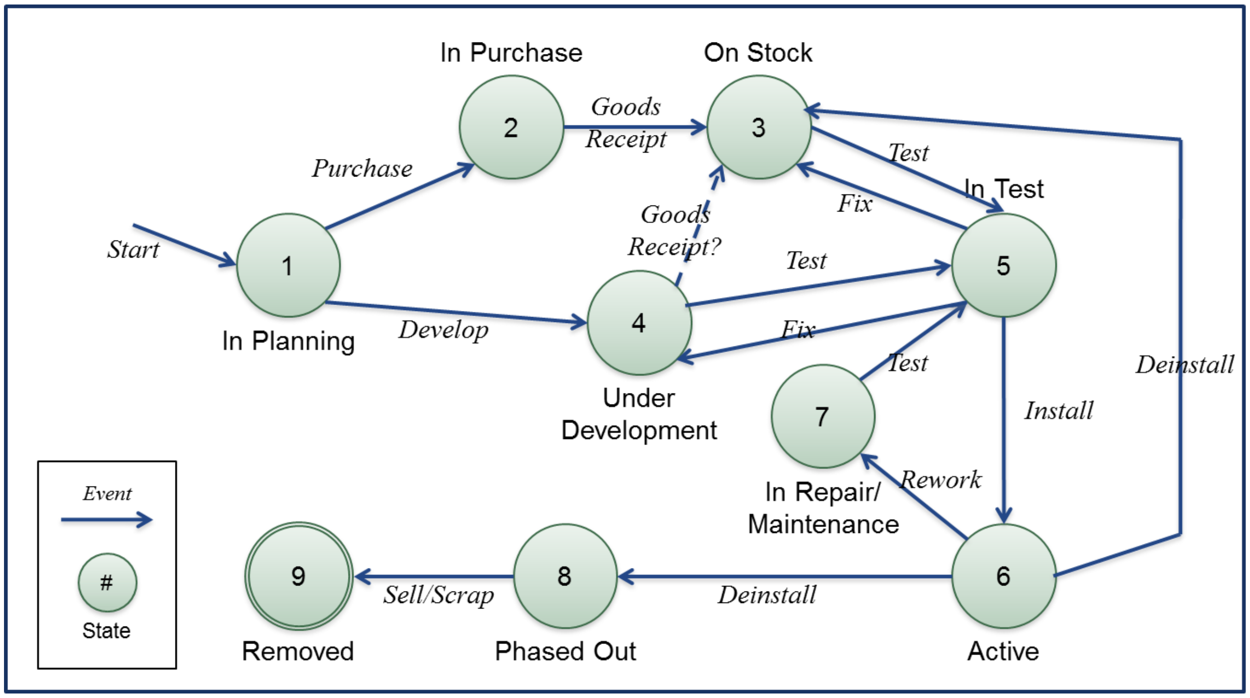


Figure 3: Lifecycle States: *Software and hardware CIs are always in one of the states 1-9, from being purchased or developed to a testing phase, and active phase after successful testing and installation, ending with deinstallation and removal from site by selling or scrapping the CI.*

C. Automatic Configuration Item Discovery

One can easily imagine that it is a time-consuming and error-prone process to try and model the relationship between software and hardware CIs manually. The CMS Tool allows to manually add relationships, even to select multiple CIs and connect them to one CI, or the other way around, but in the case of ~1500 software packages that comprise a standard SUSE Linux Enterprise Server (SLES) operating system, it will be time consuming even to manually select all those software packages from a list, let alone to figure out which few of the thousands of software packages are or are not installed on the hardware CI in question. Therefore, manual input of the configuration status will always be limited to a rather high level of abstraction. If the information about the installation of a new software CI on one or more hardware CIs will only reach the configuration manager via the GITS ECR, she has no means of controlling or verifying the actual configuration, therefore, discrepancies between the configuration modeled in the CMS and the reality will arise quickly.

A smarter approach would be for the hardware CI to inform the CMS which software CIs it has installed. Errors could be discovered much quicker that way, i.e., if one of the hardware CIs was forgotten in the installation process, or if the software CI was installed on a wrong hardware CI. The information from the hardware CI even allows for an automatic verification of the ECR that the software CI in question, in the correct version, has actually been installed on the correct hardware CIs (and optionally, that the previous version of the software CI has been removed). This idea is not a new one, in fact, the HP uCDMB system in use at GSOC comes with an already implemented *Autodiscovery* function, using probes in different network segments which, given the correct credentials, collect data via a number of network protocols and inventory the installed software. However, GSOC has a concentrical network structure, where the network areas are separated by firewalls which only allow unidirectional information flow in the outward direction. Even if a network probe were allowed in the operational area (which it is not), the instructions that would have to be sent to the probe from the main CMS application in the office area could not pass through the firewall in the inward direction. Therefore, the out-of-the-box Autodiscovery provided by HP cannot be implemented in our control center due to security reasons. However, we have devised a workaround that transfers the configuration information across network boundaries via file transfers and then makes use of the built-in HP reconciliation engine. Our method of implementing agent-based Autodiscovery in this secure network setting, and in the described multi-mission context, in order to reap the benefits of automatic CI discovery is described in this paper.

II. Autodiscovery of Configuration Items

Several actions must be combined in order to automate the complete cycle of configuration management as described in Section IA, most importantly the automation of configuration deployment and the automation of configuration verification. The Autodiscovery process is described in detail in this section.

A. Automatic Configuration Deployment

Any automatic configuration management approach that makes use of scanning a piece of hardware and returning a list of installed software CIs requires a concise definition of a *software package*. Typically, scanning inventory agents read data from the Windows Registry and/or the software listed in the Control Panel under “Programs”. For Linux machines, the RPM package manager[5] is widely distributed and also a part of the SUSE Linux Enterprise Server used at GSOC. For software installation and maintenance, the use of package management tools rather than building packages manually has advantages such as simplicity and consistency. Package management is a prerequisite for the automation of build and deployment processes. If a piece of software is installed without the use of a RPM package, or without an installer that enters data into the Windows registry, this piece of software cannot be retrieved by a scanning application. Automatic deployment of the software package via installation servers is also impossible without software packaging. In a control center system architecture, where both physical hardware and virtual machines are usually designed redundantly, resulting in many hosts which should have an identical or similar configuration, installation servers present an obvious advantage over installing and configuring a collection of hardware or virtual hosts manually and one by one. RPM package management offers the additional advantage of dependency tracking, as RPM management tools track how software packages rely on other software packages, e.g., Perl or Python libraries for tools which were written in one of those languages. At GSOC, we already have automatic configuration deployment tools implemented and strive to gradually apply them to all the missions. For the Autodiscovery pilot missions, automatic configuration deployment was used for nearly all the hosts after the early testing phase. This implies that mission-critical software packages developed in-house are required to be delivered within an installer package, which creates some additional effort for the software developers, but also enables them to observe proper release management.
Another advantage of the automatic configuration deployment concerns the deployment of the Inventory agent itself: by packaging it into an RPM of its own, or a MSI installer, respectively, it was possible to deploy the Inventory agent and the Linux cron job or Windows scheduled task, calling it once a night, onto all the hosts with minimal effort. More details on the installation server tools can be found in Ref. 2 and Ref. 3.

B. Automatic Configuration Verification

Configuration verification, as defined above, is the activity of checking if the configuration reflected in the CMS concurs with what is actually installed and configured on the configuration items, e.g., that the correct versions of the software CIs are installed on the correct hardware CIs, and that *only* the correct versions of the software CIs are installed there. The only way to manually verify the configuration is to log into each machine and compare the installed software CIs to data from the CMS. Since the operational machines are situated in a different logical network area than the CMS database, it might be practical to use a Configuration Item Data List (CIDL) from the CMS and compare the listed CIs to the reality. Any discrepancies, e.g., software CIs in different versions than they should have according to the CIDL, must then be noted in the list and updated manually in the CMS at a later point. This was the observed practice at GSOC prior to the introduction of the automatic discovery. If software CIs are found on the machine that are not in the CIDL, and hence should not be part of the configuration, those software CIs must be noted as well, and it must be researched if they should be added to the configuration, because they are required for some reason, or if they should be deleted on all machines, in the case of potentially harmful software that was installed without permission. The deletion of such software again has to be done manually, if no installation server is used. Therefore, automatic configuration verification is what we strive to achieve with the implementation of the Autodiscovery process.

[5] https://en.wikipedia.org/wiki/RPM_Package_Manager

C. Autodiscovery Process and Methods

Both configuration deployment and configuration verification could be automated on their own, but if both activities are combined, one can make use of the full cycle and verify that the automatically deployed configuration is indeed the correct configuration, and, to a certain extent, that the deployed configuration has not been tampered with.
The method that we have chosen to roll out the Autodiscovery on all relevant parts of the pilot project missions comprises the following steps:

1. *Create Mission FTP User:* An FTP user specific to the mission is created. The mission FTP user and its credentials are used to transfer the scanned configuration data of the respective mission, such that the need-to-know principle is observed.
2. *Implement Installation Servers for Mission Software Package Deployment:* Automatic configuration tools (Puppet and OPSI) are used for structured configuration deployment of software packages on multiple hardware items within each mission.
3. *Deploy Inventory Agent and Daily Inventory Job:* Installation servers are also used to deploy the OCS Inventory NG Agent software package.
4. *Transfer the XML Files:* Copy the XML Files from the OPS FTP server across segmented network boundaries to the Office FTP, from where they are in turn collected by the CMS Server.
5. *Transform the XML Files:* Run the XML Files through an XML schema transformation such that the end XML format matches the format used for XML imports into the HP uCMDB.
6. *Import Configuration Data into CMS:* Import the XML files into the CMS, where they are converted to configuration items and relationships between the items.
7. *Set the CI Lifecycle States:* Intelligent rules in the CMDB automatically set the lifecycle states of configuration items depending on the scan state to account for removed CIs.
8. *Automatic Configuration Reporting and Status Accounting:* Intelligent reports alert the configuration managers to changes in the configuration as deployed in the system, i.e., new CIs in the DISCOVERY Subsystem are shown.

Figure 4 shows an overview of the process steps, which are detailed in the following sections.

1. *Create Mission FTP User*

A mission-specific FTP user and its credentials are used to transfer the scanned configuration data of each mission, such that the need-to-know principle is observed and only persons with access rights for each mission may see the data contained in the inventory files collected in the Autodiscovery process. The mission specific FTP user and its password need to be specified in the mission-specific configuration file for the Linux cron job or Windows scheduled task that runs the inventory agent every night and transfers the XML files to the OPS FTP, cf. Subsection 3. In

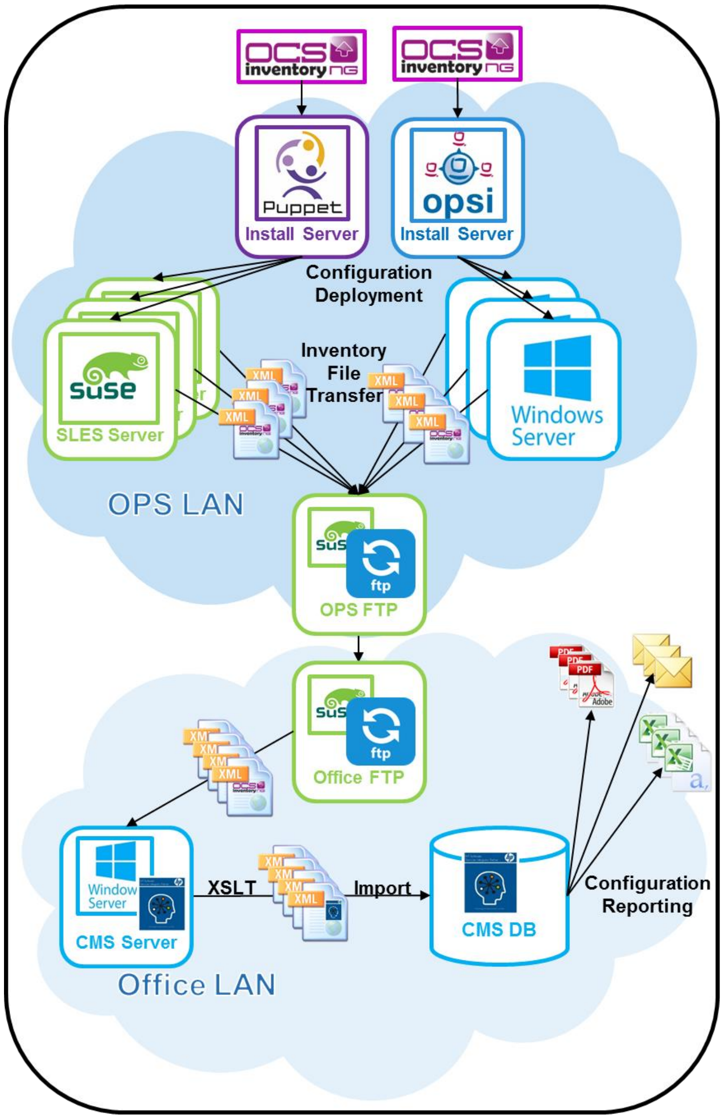


Figure 4: Autodiscovery Process and Methods: *Configuration Deployment via Installation Servers, Inventory File Transfer, File Transformation and Import into the CMS followed by automated Configuration Reporting and Status Accounting.*

addition to the mission FTP user, new missions, or missions which wish to start deploying the Autodiscovery while already operational, should create an ECR listing in detail all the hosts on which the Autodiscovery is to be deployed. A new mission whose CIs are still in the "In Planning" state shall make use of its design documentation to provide the host list. The list will ensure that no host is forgotten in the deployment, and verify that the Autodiscovery data arrive in the CMS from all relevant CIs.

2. Implement Installation Servers for Mission Software Package Deployment

The automatic configuration deployment tools already in use at GSOC are Puppet for Linux (SLES 11 in our case) and OPSI for Windows. They are used for structured configuration deployment of software packages on multiple hosts (both physical and virtual machines) within each mission. This ensures that the software packages including their configuration information (mainly their version numbers), can be deployed in a remote and automated manner, and that those software packages are registered in the Linux RPM Manager or the Windows Registry. Thus, the software packages are installed in a format that can be recognized by the inventory agent. It also allows us to roll out both the inventory agent and the agent job to run it every night in an efficient risk-reduced manner. Of course, both the inventory agent software package and the agent job software package will be listed in the CMS in their appropriate version, as installed on each host, after the Autodiscovery process has been completed.

3. Deploy Inventory Agent and Daily XML File Creation Job

OCS Inventory NG[6] is an Open Source inventory discovery agent available online. The agent should be configured to store its scanning result in a local XML file with extension ".ocs" or ".xml", which can then be shipped to the uCMDB. We use the established installation servers for the mission to deploy the OCS Inventory NG Agent software package and the daily Linux cron job or Windows scheduled task that runs the agent, creates the inventory XML file, and uploads the scan file via FTP to the OPS FTP server. The mission specific configuration file containing the mission name, mission FTP user name and its password, is also rolled out automatically by the installation servers. The configuration file is placed in the root's home directory for Linux and in the Administrator's "MyDocuments" folder for Windows, with privileges set appropriately in order to ensure that the password in the file is protected against unauthorized access.

4. Transfer the XML Files

An automatic FTP Transfer from OPS-Lan to Office LAN has been set up for transferring configuration XML files to the uCMDB server. The FTP servers can be reached via "ftp ftp.nos.office" from the Office Lan and via "ftp ftp.nos.ops" from the OPS LAN. All files placed onto the OPS FTP will be transferred within minutes to the Office FTP server. From the Office FTP, the files are collected to the CMS server once a night by a batch job. The job stores the inventory XML files (with file extension .ocs for the OCS Inventory NG Agent) in mission-specific directories and deletes the files on the Office FTP server afterwards. The process is also implemented on the CMS test server, with the exception that the batch job on the test system does not delete the XML files, as they need to be collected by the operative CMS server first, which is in charge of deletion.

5. Transform the XML Files

XML files created by OCS Inventory NG cannot be imported directly into the uCMDB. An XSLT transformation has been prepared to convert the format of the incoming files to the format recognized by the XML import job of the uCMDB. The transformation schema can be fed to the Saxon XSLT processor[7] for transformation of an input file. This conversion is wrapped in a Windows batch file, which reads all files with extension ".ocs" and ".xml" in the FTP download directories and places the resulting files in the input directory for the next stage.
During the transformation, additional CIs are placed in the XML file to ensure proper connection of the scanned CI to the existing infrastructure in the uCMDB, namely

- a project/mission CI, named after the directory where the inventory file is taken from,
- a subsystem CI named "<mission>_DISCOVERY" (with a short name of "DISCOVERY"),
- a CI collection named "<hostname>-<yyyy-mm-dd-hh-mm-ss>" using the hostname of the scanned host and the scan timestamp.

These CIs are linked to each other and to the host and software CIs created from the inventory file by "containment" relationships. A scheduled Windows job automatically performs the conversion task every night for all the files collected from the Office FTP server.

[6] www.ocsinventory-ng.org
[7] saxon.sourceforge.net

6. Import Configuration Data into CMS

CIs are "reconciliated" by the Reconciliation Engine of the commercial software HP uCMDB, i.e., scanned CIs with the same configuration identification (typically *name* and *version*) as existing CIs from previous scans are merged into one single CI. This automatically creates and updates configuration items in the CMDB with information found by the agent while scanning the hardware configuration items.

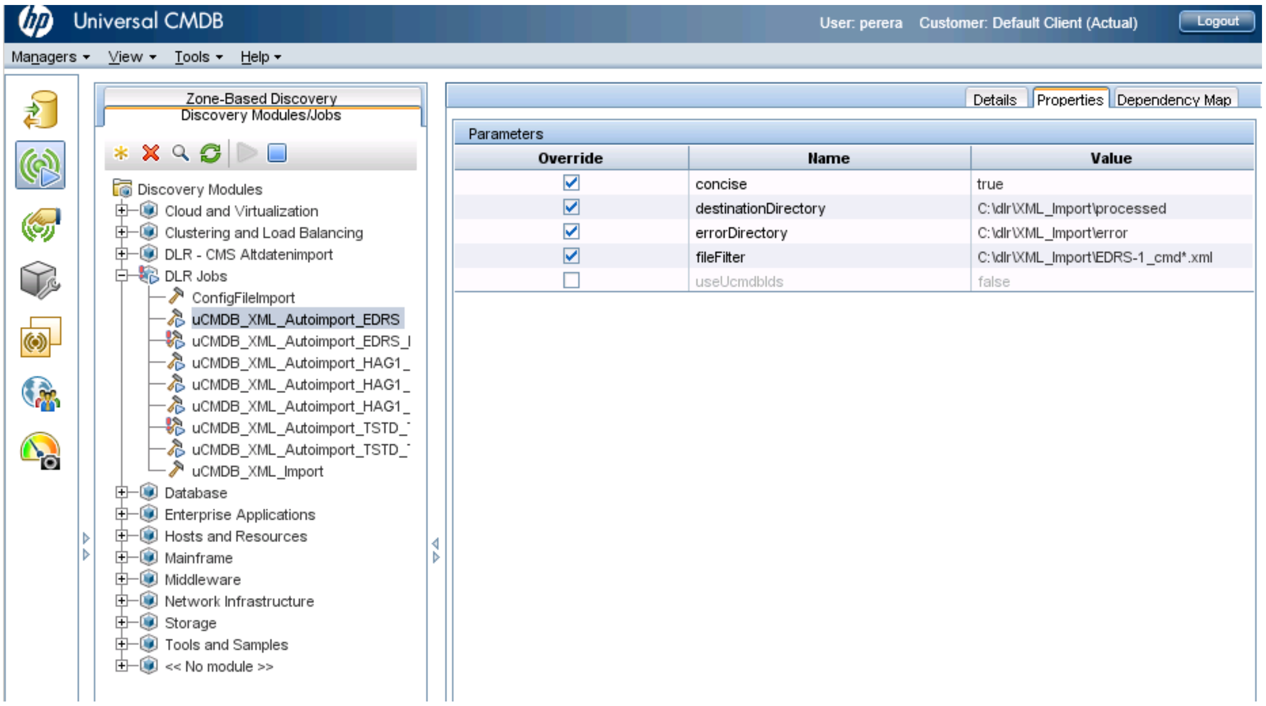
XML files formatted according to the schema used by the uCMDB Snapshot facility or a more concise format can be imported into the uCMDB using a Discovery import job, see Figure 5.

uCMDB_XML_Autoimport: a custom Discovery job based on the Jython script "uCMDB_XML_import.py" to import files from arbitrary directories on the uCMDB application server in both the snapshot and the concise formats.

As with all uCMDB Discovery processes, it is not possible to delete CIs or relationships from the uCMDB using XML imports. For example, if the installed software on a host changes, relationships to previously installed software items are not broken, and now obsolete software CIs are not deleted in the CMS. To this end, we use intelligent Lifecycle Status rules, cf. Subsection 7.

The import job displayed in Figure 5 can be controlled using the following parameter settings:

- concise: Process files in concise format, default setting: false
- destinationDirectory: Full path of the directory to move the correctly processed files to



Override	Name	Value
✓	concise	true
✓	destinationDirectory	C:\dlr\XML_Import\processed
✓	errorDirectory	C:\dlr\XML_Import\error
✓	fileFilter	C:\dlr\XML_Import\EDRS-1_cmd*.xml
	useUcmdbIds	false

Figure 5: Inventory XML File Import: *the left side shows the hierarchy of discovery modules, the right side details the settings of the uCMDB XML Autoimport of EDRS inventory XML files.*

- errorDirectory: Full path of the directory to move the files with processing errors to
- fileFilter: Full path to input files (including "*" and "?" wild cards)
- useUcmdbIds: Interpret "Id" elements/attributes as uCMDB Global IDs and use them for CI identification (this is useful for reimporting CIs previously exported using the uCMDB snapshot function or the XML export function), default setting: false

There are several Autoimport Jobs for the pilot missions where before, there was only one Autoimport Job, due to the Java heap space problems described in Section IIID. Each Autoimport Job runs at a scheduled time every night, after the previous data collection steps have been completed. After importing XML files created by the XML transformation step, CIs which had existed previously in the uCMDB will have more than one "containment" relationship: One to the CI collection to which they already belonged, and at least one to the artificial CI collection "<hostname>-<timestamp>" created by transforming and importing the inventory file. Since multiple "containment"

links are not allowed in the DLR data model, any unnecessary links will be removed by a customized uCMDB batch process (a so-called "enrichment rule"). The cleanup of unnecessary links will leave such CI collections empty which contained only pre-existing CIs (e.g. a new scan of a basically unchanged host). Any such collection will be removed by the enrichment rule "DLR_Cleanup_Discovery_Folders" which is scheduled to run once per day in the uCMDB scheduler (job schedule "CleanupDiscoveryFolders").
Both the import job and the enrichment rule cleanup jobs are scheduled in the CMS to run every night at a fixed time, just after the jobs mentioned in the previous steps.

7. Set the CI Lifecycle States
Intelligent rules in the CMDB automatically set the lifecycle states of configuration items depending on the scan state, i.e., if a software is not found on a given day on any machine belonging to the mission, the software CI lifecycle state is set to "Removed", thus hiding it from the standard views and from default CIDLs, which only show CIs in state "Active". This ensures that old versions of software CIs will not clog up the CIDLs indefinitely, but disappear once there is no single instance of them installed on some hardware belonging to the mission, just as defined in Section IB3. The Autodiscovery creates new CI instances for every software CI *version*, e.g., the OCSInventoryAgent in version 1.0.0 is a different CI than the OCSInventoryAgent in version 1.0.1 – it must be, because while Automatic Configuration Deployment would in theory allow us to update every single hardware across all the missions from version 1.0.0 to version 1.0.1 at virtually the same time, this rarely happens in practice, as each mission has its own constraints and global changes are a source of errors. Also, there are often good reasons to keep an older version of a software component active within one mission while a component is upgraded for other missions, or even to keep different versions active on different chains on the mission, e.g., the TEST chain for testing purposes, as mentioned in Section 1A. Nevertheless, it makes no sense to list all the previous version of a software CI as part of the CIDL, only the active ones should be part of the CIDL – therefore, the intelligent lifecycle rules set the status of uninstalled software CIs to removed.

8. Automatic Configuration Reporting and Status Accounting
Regular reports alert the configuration managers to changes in the configuration as deployed in the system, i.e., new CIs in the DISCOVERY Subsystem are shown. This enables the configuration managers to link the new configuration items to the corresponding change request, to verify planned changes, and to discover unplanned changes (e.g., if there is no change request for the addition of the CI in question!) Thereby, CM integration with the change management system is enforced and the deployed configuration is verified automatically.
New hardware CIs are always visible in the DISCOVERY subsystem, newly installed software CIs are only visible there if they are not installed already on some other hardware CI. If a software CI is merely added to another hardware CI in the mission, but the software CI was already part of the CIDL, the addition is merely seen in the "Related CIs View" of both the software and the hardware CI.

III. Results and Discussion

The introduction of the Autodiscovery has led to the desired effect for the affected missions; both regarding automatic configuration deployment and automatic configuration verification. Configuration managers report a much better understanding of the actually installed software packages. In several cases, human errors have been discovered, e.g., when an engineer had forgotten to install a new software package version on one of three machines, which was observed in the CMS by the configuration manager, who made the engineer realize his mistake before any errors driven by the incorrect configuration could occur in tests. This ensures accountability and traceability for all configuration items.

A. Automatic Configuration Deployment

As mentioned in Section IIA, we have already implemented automatic configuration deployment tools for the Autodiscovery pilot missions. Therefore, the mission-critical software packages developed in-house were required to provide an RPM installer package for Linux machines, or an MSI (MicroSoft Installer) or setup.exe installer for Windows machines.
The additional effort for the software developers depended on the complexity of the software package, for instance, for SCOS, our monitoring and control system, it was quite difficult to build RPM packages due to hard coded device paths and host names within the software, but our developers and engineers succeeded in the end. In the case of SCOS, a second RPM was built for the configuration files, which are extensive in number and complexity and a

known source of errors, if edited manually. The efforts undertaken to package the SCOS software and even its configuration will be well rewarded in terms of stability and failure resistance, and also enables us to observe proper release management. However, this also means that our automatic configuration management only works for our definition of software package CIs that can be recognized by the OCS Inventory Agent. Local scripts, or software binaries copied onto a machine and executed manually will not be detected by our system, only software CIs with an RPM or Registry entry will show up in the CMS. As it is forbidden to change operational configurations without an ECR, such locally copied and executed software should not exist in theory, but cannot be disproven with our approach without strict adherence to the prescribed CM processes.

B. Automatic Configuration Verification

The most obvious benefit of the daily Autodiscovery scan is that newly discovered CIs are visible within less than 24 hours in the DISCOVERY Subsystem of the corresponding mission. This enables the configuration manager to directly verify that a change has been implemented, even before the person implementing the change has set the ECR status to implemented in GITS. In the case of unplanned changes or undesired software installations, the configuration manager and the system engineer benefit from the information that new software CIs have been discovered by the Autodiscovery, and can start the investigation for the reasons much more quickly. The same is true for hardware CIs that have been forgotten in the installation process; if the CMS shows that the old software version has not been replaced on all required hardware CIs, errors are more likely to be uncovered at an early stage.
Compared to the scenario described in Section IIB, the improvement thanks to the Autodiscovery process is obvious. This enables us to actively verify configurations consistently which otherwise would just not have been manageable to verify manually.

C. Filtering: Automatic Configuration Item Data Lists vs. Manual CIDLs

As discussed in Section IB6, the use of the Autodiscovery results in the creation of an increased amount of software CIs, ca. by a factor 50 compared to the manual software CI creation. The number of *isInstalledOn* relationships is increased by a factor larger than 100. While this is very good news, as it allows us to model many entities and relationships which would otherwise have been excluded from the CMS, there was a drawback:
The many software packages inventoried by the discovery agent for SUSE Linux (~1500), but also for Windows Server (~500) have posed a small problem for the readability of the Configuration Item Data List. This list usually contains fewer than 50 software configuration items, manually modeled after the components described in the product tree for each mission. Flooding that list of 50 specific, important and mission-critical software configuration items with an additional 1500 software items that comprise the standard installation of a SLES 11 is obviously not an option. The CIDL is often a deliverable that has to be sent to the customer for milestones and reviews, who is interested in the versions of the ~50 specific, mission-designed software items, and not in detail in all the libraries that make up the SLES 11. These libraries in their various versions, as installed on different hardware CIs, may however be very interesting and helpful to the engineers for error analysis. Hence, it would be unwise to just discard the information. Our preferred solution was to group the software packages comprising a standard SLES 11 into a software CI Collection under the multi-mission project and link the top-level CI collection under all the missions which use that operating system on at least one host CI. The same was done with the Windows packages of the hosts running Windows Server. This way, each host in each project is still connected to the up to 1500 software packages that it has installed, and this information can be accessed either starting from the host end of the relationship or from the software package end, and is very useful in error analysis. On the other hand, the mission specific CIDLs produced for the customer contain only the more prominent packages specific to the mission and relevant for the change control process, and therefore remain human readable.

D. Autodiscovery Effects on System Resources

The dramatic increase in configuration items contained in the database due to the automatic collection has led to some issues with the CMS system. The CMS application server RAM of 8 GB was sometimes not enough to correctly import all the XML files due to memory exhaustion. We have solved this problem by segmenting the XML files into missions, or parts of missions, and only processing the files from one mission, one mission after the other, thus restricting the import to fewer than 15 files at a time, cf. Figure 5.
The CMS database server's hard drive is filling up much more quickly than before, due to the nightly changes that add to the data in the history databases, even if nothing actually changes. We have temporarily solved this problem

by storing a smaller amount of history data, but we shall have to increase the hard drive volume soon, especially with new missions requiring the Autodiscovery.
In addition, there was an issue with the FTP transfer: as all the machines in the OPS LAN were set to scan their inventory and upload it to the OPS FTP server in the same second, this sometimes resulted in a problem similar to a denial of service attack for the OPS FTP server. This issue was solved by randomly assigning different transfer times to the local clients, such that the same machine always transfers its inventory file at the same second every night, but the other clients do so a few seconds later or earlier.

IV. Conclusion and Outlook

In this paper, we have discussed how we have reaped the expected benefits while addressing the arising challenges in the process of successfully implementing Autodiscovery of configuration items. The automation of both configuration deployment and configuration verification enables the Configuration Managers to observe the full configuration management cycle and verify that the auto-deployed configuration is indeed the correct one, and that it has not been changed by errors in the deployment process, or manually. In addition, the importing of CI data from the OCS Inventory Agent facilitates the quick and detailed collection of CI data that could never have been managed manually.
Our method renders manual life cycle changes and manual creation of newly installed software items unnecessary, thus saving substantial time and effort. It also enables the verification of installed software configuration items (uncovering both false positives and true negatives) without additional effort. By integrating the change management and issue tracking tool with the CMS, the issues affecting, and changes effected on a configuration item during its lifecycle are recorded and traceable in both directions.
Future steps include the deployment of the Autodiscovery to all other GSOC missions, if possible. Thanks to the configuration deployment automation, it shall be relatively effortless to extend the Autodiscovery to more missions, provided they are running the same operating systems as the pilot missions.
The Columbus project, currently the only manned mission at GSOC and a large project in terms of configuration items, complexity, and personnel, would benefit greatly from the Autodiscovery. However, as the Columbus mission is contractually required not to upgrade the operating system from SLES 10 for many hosts, this requires building an OCS Inventory Agent RPM package for SLES 10 first.
In addition, the HP uCMDB allows for the creation of gold standards, against which configuration baselines may be checked automatically, and even impact analyses, given that the CIs are modeled in a cluster hierarchy in the CMS, which is not yet the case, but facilitated by the Autodiscovery. Those two features would be very interesting to apply to GSOC's missions in the future.

Acknowledgments

I would like to thank Dr. Thomas Bassler, Dr. Thorsten Beck, Tarsicio Lopez Delgado, Manfred Schmid, and Steffen Zimmermann for helping with the implementation and the infrastructure needed to make the Autodiscovery work. Séverine Bernonville and Julia Hachmann deserve my thanks for continuously supporting and testing the process, working with the autodiscovered CIs and providing constructive criticism. I thank Dr. Thomas Bassler and Friedrich Gruschke for proof-reading this paper, while I would like to thank Dr. Josef Friedrich for his constant support and helpful knowledge, for being a general joy to work with and for proof-reading this paper.

References

[1]ECSS, ESA-ESTEC, *ECSS-M-ST-40 C, Configuration and information management*, Third issue Revision 1, 06 March 2009, Noordwijk, The Netherlands.

[2]Beck, T., Schmidhuber, M, and Scharringhausen, J., *Automation of Complex Operational Scenarios - Providing 24/7 Inter-Satellite Links with EDRS*, Daejeon, Korea: submitted for publication, 2016. Proceedings of the 16th International Conference on Space Operations, May 2016.

[3]Lopez Delgado, T., *Requirements, Configuration and Testing: Improved Management through Semi-automatic Processes*, Pasadena, California: 2014. Proceedings of the 15th International Conference on Space Operations, May 2014.