Anomalizer

Optimized aggregation of anomaly indicators for spacecraft control systems

Nuno Brito[1], Rui Figueiredo[2] and José Feiteirinha[3]
European Space Operations Centre, European Space Agency

A significant part of the development lifecycle in software intensive spacecraft control systems is dedicated to correction of defects and anomalies. Even though human operators are proficient in gathering fault details, faulty indicators collected manually are difficult to isolate and lack the level of detail/precision required for support engineers to replicate the defect or correctly identify causes. Observing the current situation at ESOC/ESA, the motivation of this project is to monitor and analyze key indicators such as system logs, application logs, hardware conditions and threads in execution across multiple machines. Using industry tested data mining concepts from telecom operators, we provide metrics regarding defect incidence, defect replication and expose their correlation to specific factors. Generated data is processed and summarized to the end user through a web interface that is safely accessed from a DMZ service, reachable to authorized visitors and smartphone devices. The quality of anomaly monitoring becomes optimized with the intervention of this project, hence the codename "Anomalizer".

Abbreviations

ESA	=	European Space Agency
ESOC	=	European Space Operations Centre
DMZ	=	Demilitarized Zone
MCS	=	Mission Control Systems
M&C	=	Monitoring and Control
SCOS	=	Satellite Control and Operation System
SNMP	=	Simple Network Management Protocol

I. Introduction

THIS project results from the observation over the past 5 years that software current employed at ESOC/ESA for the monitoring and reaction to anomalies in Mission Control Systems such as those addressed by the SMON[1] project still have room for improvement. We decided to provide a prototype showcasing new features such as the introduction of a web based user interface for management, a scripting environment for tailoring of the platform by staff operators and the interaction with other tools already available on the MCS ecosystem.

The Anomalizer project was developed over the course of 2011 with the input and requirements elicitation from staff members responsible for the management of present spacecraft missions at ESOC, along with the discussion of different proposals for the implementation approach. One of the concerns consistently present in our research study is the desire for a better flexibility in the monitoring software that can meet the mission requirements and integration with other tools. At the current state of development, we have addressed most of the problems that were identified and provide a tool ready for monitoring a mission control system.

[1] Software Engineering Trainee, nuno.brito@esa.int
[2] Software Engineer, jose.luis.feiteirinha@esa.int
[3] Network Engineer trainee, rui.figueiredo@esa.int

In terms of improvements in technology usability, we adopted a web interface and worked to reduce the learning gap between technicians and developers to ease the understanding and use of the system. For example, Java language combined with dynamic scripts allowed us to merge configuration with implementation seamlessly, turning users into developers and permitting full reconfiguration of the available applications without requiring a dedicated development environment.

Over the course of this paper we will present the project details, the problem we intend to solve and our expectations towards the future. At the time of writing, our development effort continues to progress as new feedback and improvement takes place. We hope this paper can provide a concise description of our past steps, our direction and goals we aim to achieve in a near future.

II. Anomalies in Mission Control Systems

Anomalies can be summarized as *"patterns in data that do not conform to a well-defined notion of normal behavior"* (Varun Chandola, 2009), translating directly into our context in the space industry as an event that occurs outside the range of nominal operations related to a spacecraft mission. At ESOC, such anomalies are categorized by criticality and urgency. Starting from informative (not critical/not urgent), to warning and finally to fatal (critical/urgent levels) that require immediate action. Human operators responsible for spacecraft control missions are required to be aware of possible anomalies in their infrastructure and document the procedures to handle them accordingly in case of reoccurrence.

However reacting to anomalies is still one step behind predicting them, which can often be done by looking at performance indicators and patterns that precede anomalous behaviors. Long term Spacecraft missions are highly affected by eroding conditions, as the hardware, operating systems and occasionally even software requirements (e.g. Mission Planning software requirements are usually dependent on changing ground station behaviors), evolve away from what had been tested and validated at the time of the mission launch.

When a mission starts, new tasks related to monitoring of anomalies are mostly handled manually. Anomalies that had not been predicted are handled by experts as new errors require investigation. Progressively throughout the lifetime of a spacecraft mission we can typically identify recurring anomalies in the infrastructure, monitor and handle them with resort to minimal human supervision, preferably through the use of automated mechanisms and documented procedures.

Given the tendency to reduce the level of intervention by human operators during the span of a mission control system lifecycle, the mission control staff becomes progressively pressured to continuously monitor the correct functioning of the infrastructure with decreased availability of resources. Additionally, customization of existent monitoring products is a complex operation that restricts the use of this option to leverage the increase of parameters in need of monitoring.

In addition to changing requirements in terms of monitoring new and/or unexpected anomalies, users are left with the development of in-house ad-hoc methods to solve these gaps since the available mission budget is often no longer providing resources to redesign software implementations.

Altogether this implicates that fixing, documenting and investigating Mission Control System anomalies becomes a full time job with a very specific skill-set. We argue that the efforts spent in monitoring Mission Control Systems can be greatly reduced for both current and future missions via the adoption of our tool, Anomalizer.

III. State of the art in ESOC

In ESOC, spacecraft missions such as Cryosat-2, GOCE, Bepi Colombo, SWARM depend on tools such as MCCM or SMON-II to monitor and notify of software anomalies in Mission Control Systems. Each of these systems has its own advantages (for example MCCM is a light weight tool, SMON is built on top of the SNMP with network monitoring) and disadvantages (MCCM is outdated and runs solely on Solaris, SMON is not scalable since it fails to track more than 10 machines satisfactorily).

Given the present usage of these tools in operational missions, José Feiteirinha in the role of control systems specialist engineer has identified two points of usage that could be improved over the current approach:

1. Integration with other tools in use by the mission
2. Effortless customization by the mission control team (to new unexpected scenarios that arise during routine operations).

To solve these two points it is necessary for engineers to develop in-house solutions that address newly found needs and so came to existence the proposal of rethinking the current methodology, consolidating onto a single system a platform that can be reused, expanded and improved by the staff in charge of mission control systems.

IV. COTS alternatives

At the moment, one of the most popular solutions in the IT monitoring market is Nagios[2]. Initially released in 1999[3], Nagios features the monitoring of network services and protocols (our requirements specified SNMP, ICMP and SSH), along with host-specific attributes such as disk-space and CPU/RAM usage. Other noteworthy features include a scalable/easy to use web interface and the ability to configure pager/SMS notifications to users.

Although Nagios users include Thales, Zodiac Aerospace, T-Systems, CERN, Philips, Siemens, Toshiba, among others[4567] and this level of adoption gathered our early interest, we eventually came to the conclusion that this tool was not the most suitable option for our particular needs. The main reason being that Nagios requires additional hardware/management with potential logistic implications, while our goal is to provide a simple tool to monitor a Mission Control System, using the existing Hardware and Operating System currently in use by ESOC and associated contractor companies.

For the intended purposes at ESOC, the most attractive functionalities from Nagios were syntax simplicity of the user configuration with an intuitive easy to use interface. Currently, Nagios does not provide the artificial intelligence capable of performing data-mining operations on the obtained failures and provide a clear answer in regards to the cause for the failure, leaving this interpretation effort for the system administrators.

Another requirement not met with Nagios is the ability to provide the system integrators with an API that would use an industry standard interface for simplicity of use. This API should be portable, to help us eliminate complex dependency problems that might arise in the future. Since Nagios is built on top of the C programming language, this would become an issue when considering the myriad of platforms where it would be running.

An additional problem with using Nagios would be the lack of control over the software, since it is owned by a foreign entity. Even though the GPLv2 licensing scheme allows a margin to fork the project to suit the Agency's needs (as was already the case before[89]), the fact is that the Nagios team could one day head in a direction not aligned with our own needs. If such a scenario were to take place, instead of a small team configuring a select few files for a proven and stable platform, we would end up with the need to re-engineer a large project. Possibly at a stage of a mission where funding for this kind of development is no longer available.

V. The Anomalizer approach

Presently, the resolution and detection of new anomalies requires that engineers implement monitoring scripts that run isolated on the affected infrastructure component. These scripts generate logs at scheduled intervals of time that are opportunistically analyzed by the staff engineer for discrepancies.

Our approach enables the currently existent monitoring scripts to be incorporated into the Anomalizer Framework, offering features such as data correlation, generation of graphics and system overview integrated with a user friendly interface that uses simple menus (e.g. when a task returns 'Error', it is automatically re-colored in red by the Anomalizer Framework GUI).

BASE PLATFORM

The first decision was the choice of base platform. We chose Java as the language and runtime platform to ensure that our project would run isolated from the underlying operating system and CPU architecture. It was successfully tested that our project could run in Windows XP/7 during our development phase and run under the Linux and Solaris systems when deployed in production mode.

USER INTERFACE

To maintain consistency from a usability perspective, we adopted a web based user interface instead of the fat-client approach as seen on SMON, this removes the need for physically installing client software on monitored machines, while providing access to a multiple user and device paradigm. This solution eases the deployment of new versions and maintenance as only the web server side needs to be modified and deployed.

SCRIPTING SUPPORT

When discussing a new approach, we had the goal of replicating the current work flow as much as possible, keeping in mind that an engineer will often write his own scripts to automate monitoring tasks when necessary. Therefore the new platform should allow engineers to work, detect and solve anomalies in a similar way to their current practice today. As a solution, we incorporated support for BeanShell inside the project to ensure that our monitoring logic could be rewritten and modified without constraints.

SIMPLICITY

Last but not least on our list of requirements is simplicity. When looking at the current monitoring tools for the infrastructure, engineers are faced with proprietary interfaces that are not documented, along with a software architecture that is not intuitive to grasp. In this project, it was our goal to preserve simplicity in design. Providing a solution that other engineers can understand and modify by themselves without requiring a steep learning curve.

With these points in mind, we progressed onto the implementation stage of this project.

VI. Implementation

In order to simplify development and reduce the implementation effort, we applied COTS whenever the licensing terms allowed us to use them freely. As mentioned in the previous chapter, we adopted Java as the base language and platform for development. This design choice brings the advantage of abstraction from the hardware and Operating System layer. In addition, we open access for other developers to participate as the Java language is used extensively across the ESA domain.

Remedium[10] is an Open Source project that handles distributed databases in an innovative way to target security flaws and was adapted for use as our base for development. By working on top of Java, it provides access to HSQLDB, Graph generation libraries, web interface and a structure for managing user accounts. This choice allowed focusing the bulk of the coding effort on the application itself, taking advantage of the libraries already tested and made available beforehand.

During the architecture design discussions, we wanted to keep the system as open to future improvements as possible. One of the decisions was providing an application that would not be statically compiled but rather scripted. The rationale for this decision is allowing new user scenarios to be incorporated onto this project with shorter time intervals for implementation, providing the means for quick prototyping and deployment of new versions, appropriate features to address a scenario of continuously changing requirements as necessary.

To achieve this goal, we selected BeanShell, a Java-like scripting language that provides dynamic interpretation of scripts at runtime. This allows the application to be scripted in a way similar to how current web frameworks based on the PHP language operate. We opted for BeanShell since the programming paradigm is inspired by Java language and can interact directly with the host Java Virtual Machine (JVM). Instead of adopting a scripting language that would be new to infrastructure engineers, we decided to adopt a language that would not be difficult to understand by third-party developers. On top of these features, the scripting language is a mature product with over a decade of existence and adopted by software products such as OpenOffice.

While creating a table with the desired requirements and sketching the effort required to implement them, it became clear that defining a strict architecture early in the project inception stage would not be a good design choice. At this point of development we had a notion of the results we desired to achieve but there were multiple design approaches that needed to be evaluated and decided upon.

Therefore, we avoided rigid decisions about design and focused on prototypes that could bring us more insight on the what we could/should implement. We invested our initial effort in creating a prototype to test different approaches until our results could be reached with satisfaction.

With this methodology, we successfully reduced the communication gap between developers and end-users. In possession of a prototype available for feedback, we were able to discover which properties of the application that required changes or improvements. In this manner it was possible to iteratively reach a design based on direct feedback from users along with an effective exploration of the available resources.

In terms of IDE (Integrated Development Environment) we resorted to NetBeans. As depicted on Figure 1, this is the officially supported IDE by Remedium and provides an integrated solution for scripting the code in the Java and Beanshell languages.

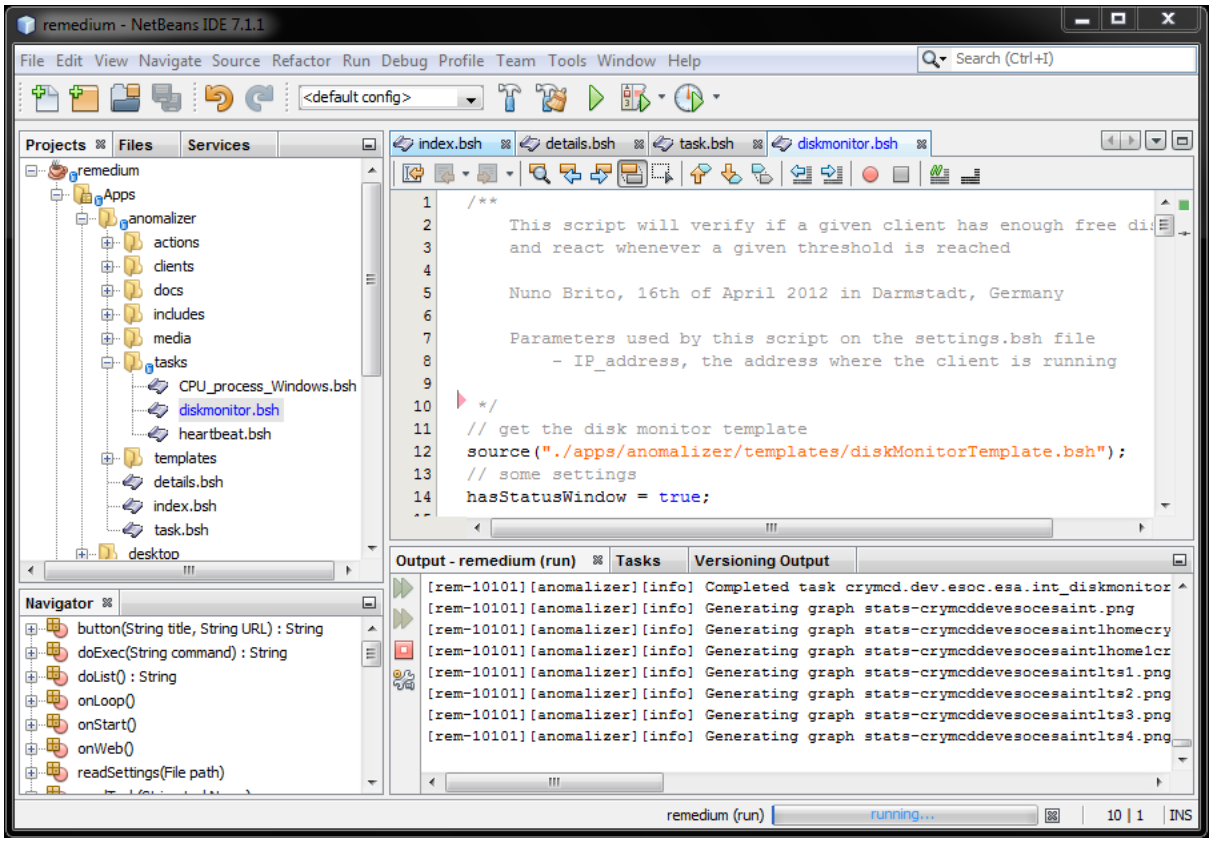

Figure 1

On this specific IDE an SVN client is available by default, supporting the goal of keeping inside one single IDE the whole toolkit required to apply, share and track developments on this project.

VII. Architectural overview diagram

When designing this system, we defined a simple diagram as depicted on Figure 2. We identified our potential end-users accessing this service from the office workstations and laptops, along with potential VPN access through personal computers and smartphones from the Internet.

Inside the server where the Anomalizer platform is running, we ensured that management actions can be performed through a web interface where the end-user can see the current status of all running tasks, along with their management. In the design stage, we made arrangements to ensure that a Command Line Interface would also become available for end-users accessing the server machine through SSH.

On given intervals of time, Anomalizer will be generating reports and graphs in regards to the status of all running tasks and store them on the file system. Each of the ongoing tasks contact the monitored items and then stores the results inside the database.

In case one or more of the monitored items does not bring an expected result, an alarm is triggered and a reaction from the ongoing task is launched.

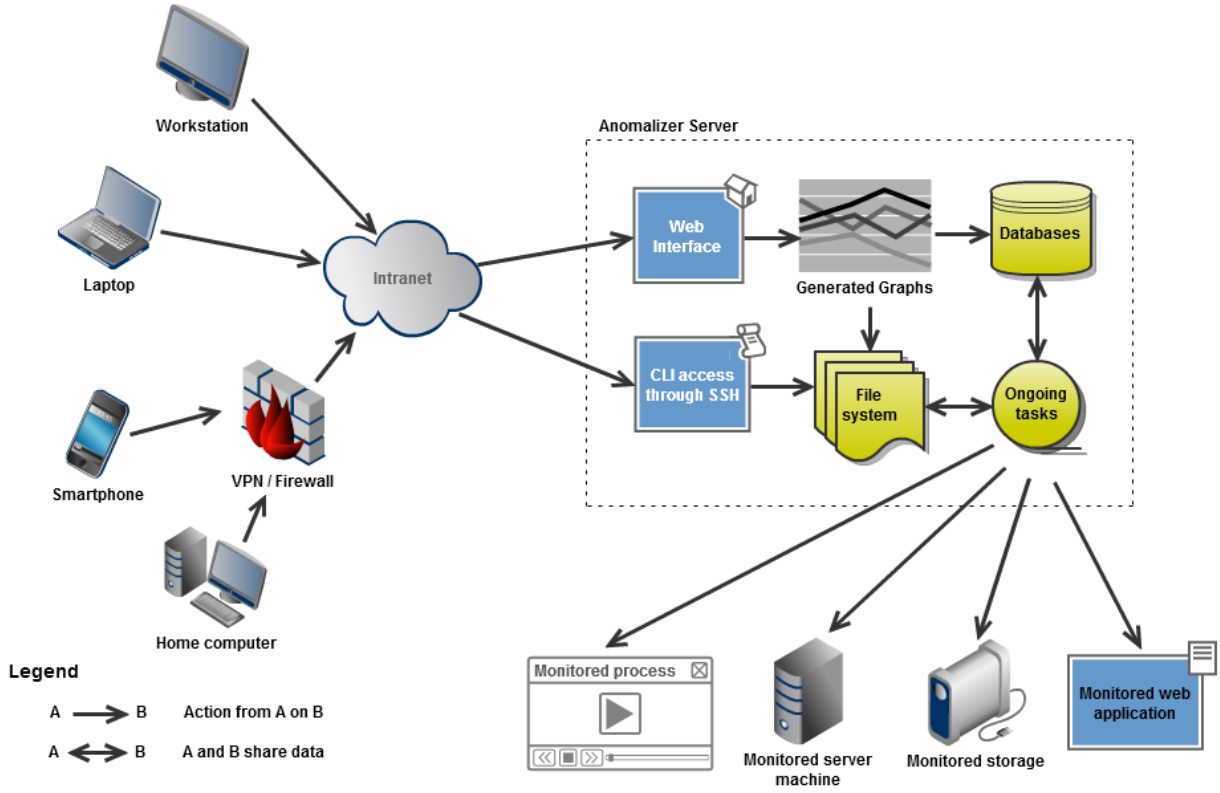

Figure 2

In Figure 2 we depict the business logic of this application, contacting each of the monitored items that can assume the form of a process running on a remote machine, the keep-alive monitoring of a server, the verification if there is enough disk space on the partition of a remote machine or checking the availability of a web page across the day.

From the web interface the end user can see a report and graphs that are created with resort to the database. We ensure that these graphs are cached on the file system and in this way improve the user experience in terms of speed performance when visiting the web page where graphs are displayed.

The Web Interface access is described in the “Usability Design” chapter, the CLI (Command Line Interface) is described in more detail in the “Command Line Interface” chapter. The role of “Ongoing tasks” is detailed in the “Monitoring tasks” chapter.

VIII. Monitoring tasks

Tasks run by the Anomalizer are based on Beanshell scripts. The intention is providing Anomalizer as a framework where engineers can later add tasks that perform the monitoring that is desired. This way we ensure that each mission can select and customize the monitoring tasks that suit their particular needs, while still using a common platform. For the purposes of testing our prototype, two tasks were initially made available:

- Ping
- Diskmonitor

The first task is a ping executed from the machine where Anomalizer is hosted onto the target client. Consequently, the status of this task is a direct result of whether our task is capable of successfully contacting the remote machine or not.

The Diskmonitor task is more elaborate and intended to take advantage of the features provided by the Remedium platform. It will use the built-in SSH client to log onto the remote server client machine and gather information about the free disk space on the targeted partitions of the machine.

After gathering this information, it is stored in an HSQL database. Whenever the end-user desires to view the status of this task, it will output a graphical representation of the percentage of free disk on each partition across a given interval of time. The screenshot on Figure 4 is an example of the outputted graphic.

More tasks are scheduled to be added in the future, once the platform consolidates onto operational deployment.

IX. Usability design

As previously mentioned, we aimed to present an intuitive interface for users and decided to adopt a web based panel for displaying data and controlling the project. To keep our design as simple as possible, with two distinct views for end-users.

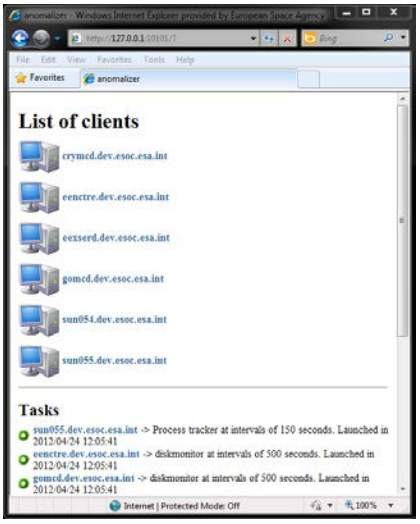

Figure 3

On Figure 3 we see the first view. This view depicts a list of clients that are available for monitoring, alongside with a complete list of tasks in progress.

Each of the listed clients includes an URL that opens the individual view, which is depicted in more detail in Figure 4.

The first/main view allows engineers to quickly visualize the machines under monitoring and evaluate the state of each task being processed.

To keep our design simple to understand, the colored circle next to each one of the tasks in progress gives a status indicator:

- Green stands for everything working as intended
- Yellow indicates that something needs attention
- Red means an error has occurred
- Grey means that the task is paused

We adopted visual cues with colors to provide a quick grasp of the overall system status without requiring much effort.

As rule of thumb, we named each tracked client according to its own DNS designation. This allows engineers to quickly recognize the mission domain where each machine belongs to. At this time we opted for a design where we can comfortably display up to 30 machines on a single page, we agreed on this self-imposed limit as a way to keep the initial view as simple as possible.

The web pages of the Anomalizer are suited for browsing from a desktop web browser such as Internet Explorer, Mozilla Firefox, Safari, Google Chrome or Opera.

In addition, we tested web browsers from mobile devices powered by Android and iOS. This mobile support provides mobility for engineers to track the status of their monitored systems on the go, using the local intranet or the Internet through a VPN office connection. From our gathered feedback, this kind of mobility was particularly welcome on work meetings where a team is now capable of quickly visualizing the status of their infrastructure with their own laptop, tablet device or smartphone.

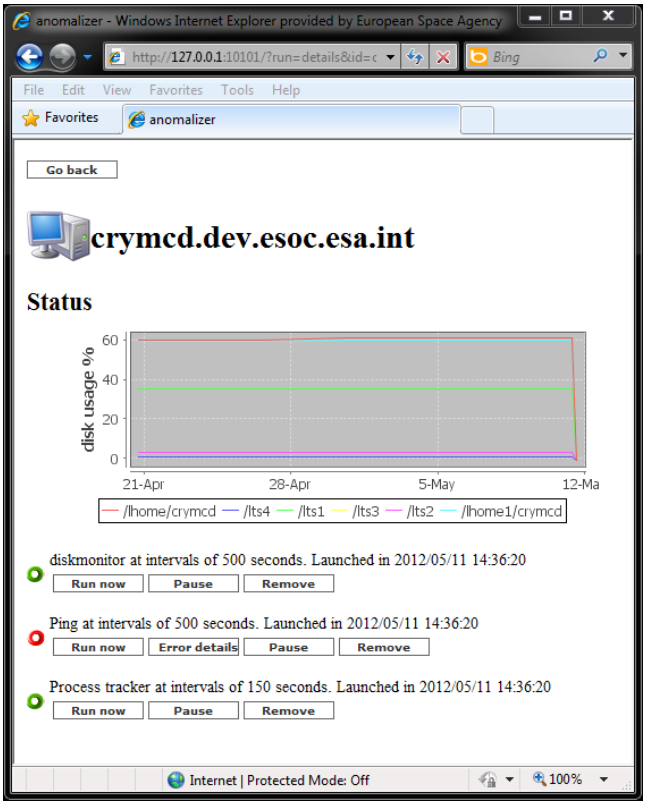

Figure 4

On Figure 4 we depict the detailed view of a client machine. In this case it is possible to view the result from the tasks that are currently running on this client such as “diskmonitor” and “ping”.

From this view, the user can pause, remove and run the currently active tasks. On the bottom of the page we see a list of available tasks that can be added to this client.

For each task running on the client it is possible to define a status window. This feature is used by diskmonitor that outputs a graph that depicts the percentage of disk usage on the monitored partition.

This graph shows the trend of disk usage across a selected interval of time and allows engineers to quickly evaluate if any action will be needed from their side in the near future.

Each task has the flexibility and means to decide which type of information should be displayed to users. On the case of diskmonitor a graph gets displayed, whereas for ping only a simple indicator of green status is sufficient.

X. Command line Interface

As depicted on Figure 2, we provide the means for end-users to control the Anomalizer with resort to SSH, this is what we call our CLI (Command Line Interface). From SSH it is possible to:

- Add / remove / rename clients
- Add / remove / edit tasks
- Start / stop the Anomalizer

We idealize the CLI as an approach for quickly changing the settings of ongoing tasks. Allowing a customization of our platform without interruption of service.

XI. Security considerations

We took advantage of the security mechanisms made available from within Remedium. This way it was possible to avoid manually implementing security mechanisms and take advantage of a framework that will continue to be improved in the future. Below is a concise description of security features that were made available:

- User authentication, we can define pages that can be viewed with permissions as “guest” or define areas that only a logged user can view
- HTTPS enabled. We have enabled the secure and encrypted transmission of data through port 443
- Roles and groups, we can define pages and functions that are only available for users belonging to a specific group. For example, only users with Admin permission can enter the Administration console
- SQL injection and XSS protection, the inputs from the application are defended from these web exploits

In the future we are considering adding more features such as DDoS protection. However given the consideration that this platform is intended to run on internal/secure networks, it was decided that this feature would be added to a later version.

XII. Field testing

Alongside with the implementation it was important to gather up to date feedback from end-users. We decided to routinely test the development version of Anomalizer on server machines powering the S2K system as a model for our testing base. We were capable of running our project alongside other monitoring tools without interfering in their activities and gathered valuable input about the work methods used by other engineers during the routine activities.

One of the added benefits of this approach is the early adjustment of this project to the real needs inside the SWARM mission control system. For example, we learned from this input that an anomaly does not always reveal itself as a drastic event that requires immediate attention, but as an event that progressively evolves to a situation where action is needed.

In face of this user scenario, we learned the importance of continuously gathering data. As result, we are now providing a graphical representation of factors such as available disk space or memory usage, allowing engineers to trace an expected evolution of usage and predict when an action needs to take place. This strategy proved to be a valuable way of discovering relevant requirements not previously noted during the project design.

The testing hardware was based on SPARC machines with Solaris 8 and x64 machines with SLES 9. Additionally, regular Windows XP/7 workstations were used as default target machines for development purposes.

XIII. Results

In the present state we have delivered a functional prototype that was tested on-site at two ongoing space missions (Cryosat-2 and GOCE). We achieved the goals proposed during the project inception and designed an architecture capable of incorporating new functionalities in the future. Furthermore, this tool has successfully replaced the need for using custom tracking scripts, consolidating on a single platform a monitoring console that now provides even more information than the status quo. The Anomalizer empowers support teams to react more promptly on time and continuously gather data about the status of their critical mission control system.

One of the most rewarding aspects of this project was the transparency provided by our monitoring process. An engineer responsible for maintaining the infrastructure can understand in detail how the monitoring and tracking are executed by our application given the simplicity of the design and reuse of processes already in practice.

Another interesting advantage is the fact that we no longer require installing any client software on the machines that are being monitored. The Anomalizer incorporates an SSH based method to gather required information to a central location. This means that we can manage monitored clients remotely and this feature eases the maintenance process for cases where a new version needs to be deployed since it becomes a question of updating a central location rather than updating the group of subordinated clients.

XIV. Future steps

The next steps is the expansion of the Anomalizer project from the specific context of Mission Control Systems to the domain of the Software Maintenance section in the division of Ground Infrastructure (HSO-GIM) products and possibly beyond. Preceding projects such as LogTracker at GIM have exposed the need for a monitoring solution capable of analyzing the internal logs and reacting to relevant events.

References

[1]SNMP based MONitoring environment
[2] Nagios, "The industry standard in IT infrastructure monitoring", http://www.nagios.org/
[3] NetSaint's changelog, http://web.archive.org/web/20060501150621/http://www.netsaint.org/changelog.php
[4] Nagios user base, http://www.nagios.com/users
[5] T-systems case study, http://www.netways.de/fileadmin/documents/Case_Studies/Success_Story_T-Systems.pdf
[6] Monitoring at Thales, http://www.netways.de/uploads/media/Pieter_van_Emmerik_Nagios_at_thales_02.pdf
[7] Central data warehouse for Grid Monitoring, CERN report, https://openlab-mu-internal.web.cern.ch/openlab-mu-internal/03_Documents/3_Technical_Documents/Technical_Reports/2011/Ioan_Gabriel_Bucur_report.pdf
[8] CNET report, Open-source working as advertised: ICINGA forks Nagios, http://news.cnet.com/8301-13505_3-10234275-16.html
[9] ICINGA, Open Source Monitoring, https://www.icinga.org/
[10] Remedium, Java Web Platform, http://reboot.pro/about/remedium