Doing the Same – But Differently
Plug & Play Solutions for Ground System Operations

Dr. Armin Hauke,[1] Erica Barkasz,[2] Marcin Gnat,[3] Dr. Udo Häring,[4] Matias Lantschner,[5] and Klaus Wiedemann[6]
German Space Operation Center, 82234 Weßling,Germany

Plug and play solutions, well established in IT-systems, may also be applied to systems used to operate satellite mission ground segments. Especially at ground stations, the exchange of antenna hardware might be necessary during the lifetime of the supported missions. We describe a possible way to implement such changes, without affecting the provided services, the established operations procedures and the qualification of the entire system. We further analyze, how an additional layer of abstraction, encapsulating the particular hardware in use with respect to the tasks to be fulfilled, can minimize the effort needed to adapt as well the ground station monitoring- and control system as the procedures to operate the newly implemented hardware. Furthermore we highlight the relation between the abstraction needed to implement plug and play solutions and the abstracted description of services defined by the Cross Support Services area of CCSDS. Finally we discuss the potential to apply the principles of plug and play solutions, developed for ground station equipment, to mission ground segments in general.

Nomenclature

M&C	=	A system to remotely monitor and control some given hardware equipment on ground.
SpACE	=	A generic M&C-framework developed at GSOC.
PDB	=	Parameter Database, internal protocol to exchange data based on a defined parameter structure within the SpACE-framework.
WARP	=	Weilheim Antenna Remote Processing, the M&C used at Weilheim ground station as antenna control.
NEMO	=	Network Monitoring, the M&C used to control the IT-infrastructure at GSOC.
Antenna	=	The sum of a physical antenna dish, its driving controls, and all the RF- and base-band equipment connect to it.

I. The Need For Plug And Play

One of the keys to improve operation of a ground segment is to become more efficient in the use of the given assets. For ground station hardware this may mean to establish pool concepts and reuse the existing hardware whenever possible for several missions. In addition one can reduce the need of having spare-parts at hand if one is able to use one given part as spare for any system, wherever it is needed. The former requires to be prepared to support any mission on any available hardware, usually named "multi-mission concept". The latter can be achieved in two ways: Either one uses equal types of equipment for all systems – which is hard to maintain for a variety of systems over a long time-span – or one is prepared to dynamically change parts of the system, without changing the performance of the system. This feature, the ability to replace some hardware by different equipment, without changing the way, the system as a whole is operated, we shall call a "plug & play solution".

[1] Deputy Head of Department "Communication and Groundstation", GSOC, DLR; armin.hauke@dlr.de
[2] Operations Manager, Ground Station Weilheim, DLR; erica.barkasz@dlr.de
[3] Ground Data System Manager, Department "Communication and Groundstation", GSOC, DLR; marcin.gnat@dlr.de
[4] Senior Software-Engineer, Department "Communication and Groundstation", GSOC, DLR; udo.haering@dlr.de
[5] System-Engineer, Ground Station Weilheim, DLR; matias.lantschner@dlr.de
[6] Senior System-Engineer, Ground Station Weilheim, DLR; klaus.wiedemann@dlr.de

A. Multi-Mission Support at Weilheim Ground Station

Weilheim ground station, located about 50 km south of Munich, Germany, performs support of satellite operations since 1968. Besides hosting dedicated antennas exclusively in use for specific projects – typically being communications satellites at geosynchronous orbits – Weilheim ground station operates six antennas in multi-mission context. There are 3 antennas for receiving and transmitting signals in S-band, two of them with 15 m in diameter, the third one with 9 m. A fourth antenna, 11 m in diameter, is prepared to receive and transmit signals in Ku-band, and a fifth of size 13 m works in Ka-band. All of them are full-motion antennas optimized to follow quickly moving targets, they are capable of receiving and transmitting signals, and all can be operated in autotrack-mode. The sixth antenna, a dish with 30 m in diameter, is designed to operate in L-, S- and X-band. This last one at the moment is equipped as a receive-only antenna.

The tasks fulfilled by those antennas range from routine TT&C-operations for LEO-satellites, primarily done in S-band, over LEOP-supports in all frequency ranges for LEO- and GEO-satellites, to IOT-campaigns. All these antennas also are used for R&D-work. Especially the largest antenna, the 30 m-dish, is utilized for various test-campaigns, because of its easy access to the feed-system and the possibility to exchange and adjust the connected RF-equipment.

Some boundary conditions for operations and the applicability of plug & play solutions can be derived from this portfolio of activities: As used for special tasks in critical mission phases, all equipment is highly redundant and the antenna hardware is designed such, that operation is still possible, even if several parts of the equipment fails. As an example, the up-link chain for transmitting signals is not only redundant in a sense of two independent existing chains. The signal path can also be crossed from one chain to the other between base-band devices and frequency-converters, as well as between frequency-converters and high-power amplifiers. Therefore the system is robust not only against single failures, but in turn, it is more complex. In consequence, when changing the particular type of one device, it has to fit well not only to the devices within the same chain, but also with almost any other equipment present.

However, the same example shows the appealing of plug & play solutions: The two 15m S-band-antennas are equipped each with three frequency-converters in the up-link, as they are also prepared to support two satellites in close formation at once, using the same dish and the same high power amplifier.[1] That means one would have to replace either 6 and more frequency-converters at once to keep all the hardware the same just at those two antennas. Alternatively one has to be prepared to work with a mix of several models.

B. The Benefit of Abstracted Task Descriptions

During the recent years, all antennas at GSOC's ground station at Weilheim have been equipped with the monitoring and control software called WARP. This software is based on the DLR in house developed M&C-framework SpACE and has been already presented.[2]

One of the major improvements with this new M&C-system was the abstracted way of describing services independent of both, the particular antenna in use and the mission to be supported. The benefit here was to standardize the actions necessary to be carried out to successfully support a mission. On one hand, we got rid of mission specific configuration effort. On the other, we achieved harmonization of routine operations in a way, that from the point of an operator controlling the support, every support does look the same. The predefined tasks to be initiated by the operator are called "workflows". It is remarkable that a common set of workflows can be utilized for the full span of supported frequency-bands (L- to Ka-band) and orbit types (LEO, GEO and even lunar orbiters).

This abstraction was realized by defining the antenna characteristics as well as the mission definition in generic terms and deduce the explicit action within the workflows as functions, taking the generic mission- and antenna-description as arguments. As successful as this first attempt to include a layer of abstraction into the definition of the workflows was, completely out of scope was a possible second layer of abstraction: the complete decoupling of the tasks from the particular hardware.

C. Abstraction versus Completeness

There are two reasons for not abstracting the particular antenna hardware already from the beginning. First, when starting to implement WARP, the hardware elements building up the whole antenna system were thought of as being kind of static, not being exchanged often. This assumption turned out to be false. Beside the potential to use existing hardware more efficiently when being able to exchange it without a big hassle, the role of Weilheim ground station as a center for research and development leads to a much more lively work on the antenna equipment than expected.

This extensive use of the existing hardware for testing and research also gives the second reason, why abstracting the hardware was not foreseen from the very beginning. Abstraction works well when similar tasks can be identified and grouped. However, the nature of research is to test and stress all the equipment to the edges of their capabilities. Routine operations – as the word routine underlines – base on recurring tasks which may be carried out on one or the other way, but resulting in the same after all. R&D-work in turn relies on the possibility to overcome this routine and try out new paths.

From the discussion above it is clear that a system engineer doing R&D-work with the ground stations hardware will necessarily need full control of all of the implemented hardware at the deepest level of detail. This requirement contradicts the attempt to bundle a subset of similar tasks within a layer of abstraction in the strongest possible way. And the dilemma can only be overcome when the basic access to each hardware device still is possible at a dedicated low level access for system engineers while on top of that the abstracted layer is used for routine operations. But this also means that there will be parts within WARP that must be changed when hardware is exchanged, even if those parts of WARP are not relevant for routine operations. Nevertheless, scenarios are feasible where routine operations immediately continues after a change of hardware, and it is tolerable to only gain the full functionality at lowest level later on.

Having this said, let us now analyze the structure of WARP and discuss where to integrate best an additional layer of hardware abstraction to move towards plug and play solutions.

II. Realizing Plug And Play

The schematic architecture of the WARP software is shown in Figure 1. WARP, as its underlying M&C framework, is a distributed system. The interfaces to the controlled hardware are called generators. These generators translate all information provided on and by the hardware into the internally used PDB protocol and forward the data to a central server. Counting the data exchange between hardware and generators in their device depending protocols as data on Level 1, this low level information converted to PDB is labeled Level 2 information.

A software part called processor (PRC) is also connected to the server (in fact there are several processors for different tasks). The processor receives all L2 information and uses this complete data set to derive higher level summary information, consequently named L3 and L4. Typically, Level 3 summarizes information from a single device, while Level 4 combines the information of a functional group of hardware, lets say an up-link chain as a whole. If needed, the layering can continue, for example providing summary information on several antennas at a station overview, that may be counted as Level 5. Whatever level it may be, the processor (PRC) acts as generator for this derived information and forwards it to the server as the generators do for L2.

From a technical point of view, differentiating between the various levels on top of L1 is purely semantics. In all cases, there is one part of WARP responsible to create this information – generators for L2, processors for L3 and L4 – and all of them forward their data to the centralized PDB-server who is responsible to distribute the information to any client, needing this information – in particular the front-end GUI-processes accessible to the user.

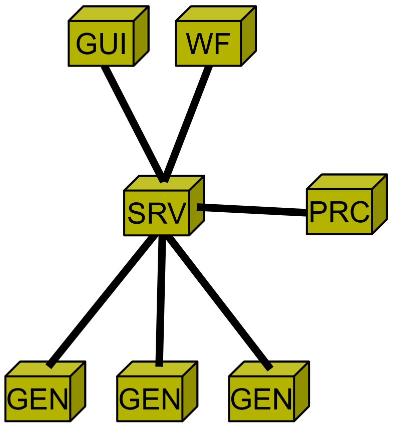

Figure 1: Schematic structure of WARP – see text for details

A. The Impact on WARP when Hardware is Exchanged

Suppose a single device at an antenna has to be replaced by new hardware. To find out, which components of WARP are needed to be adapted, we have to look for places where L2-information of this device is involved, because of L2 being the lowest level information provided by the hardware. Following the discussion above, these parts of WARP are:

- **The generator for the particular device**
  As the generator provides the interface between WARP and the device and has to connect to the hardware and receive data in a proprietary protocol defined by the hardware, there is nothing we can do but to adjust this interface if the hardware is exchanged. So worst case, if a completely new piece of hardware shall be installed, a new generator has to be developed.
  However, more likely an old device shall be replaced by some newer equipment, already tested and in use in some other place, so the generator for this hardware is already available and can be plugged into the particular WARP instance thanks to its modular design.
- **The server**

The PDB server, being the central component of WARP, is not sensible to the type and content of the information it is supposed to propagate. As long as the connected clients use the common PDB protocol, the server will handle the data correctly. Therefore there is no need to adjust the server to an exchanged hardware – with one exception. At the moment, it is part of the servers configuration, which generators are part of the WARP instance and which parameters these generators provide. Therefore, for some new hardware, the configuration of the server has to be adapted and the server has to be restarted with this new configuration. In the context of an antenna-M&C like WARP, this is absolutely feasible. The update of the configuration including the restart takes only few minutes and can easily be managed during the antenna downtime caused by the hardware exchanged itself. However, discussing plug and play solutions, this is something that one would like to avoid. Indeed having the generators and their parameter sets explicitly given in the server configuration is more a design feature than a technical requirement. Changing the servers configuration dynamically with the connection of altered or new generators could be implemented if needed.

- **The processor**
  Being the application that translates basic L2 parameters into summarized high level information, the processor naturally is a place where hardware specific data can be (re-)formulated in an abstracted way. Due to the design of WARP, the processor receives the complete set of L2-information and hence automatically gets all needed inputs. But the remaining question is: Can the needed logic inside the processor be implemented in a way that it continues to work properly without further effort, once the structure of the received L2-data does change? Answering this question is the key to make the system as ready for plug and play as possible and we will discuss this in the following.
- **The GUI**
  As discussed above, it is an external requirement set by the work performed at Weilheim ground station, that the GUI shall provide the complete set of monitoring and command capabilities for all connected hardware. It is only due to this requirement that in parallel to the exchange of hardware, the corresponding forms in the GUI have to be exchanged as well. The GUI would be independent of the controlled hardware in a natural way, if it would only display higher level information. In other words, there is no need to spend effort in the attempt to let the GUI dynamically react to changes in the displayed hardware. The GUI is inherently capable for plug and play, if its displays are limited to abstracted information provided by the processor.
- **The Workflows**
  The same way, the processor is the key component to convert hardware specific monitoring data into abstracted information, the workflows are the corresponding component for commanding. As pointed out above, the workflows (executed in a special application denoted with WF in Figure 1) do this abstraction already in terms of mission- and antenna-specifics. Introducing another layer of abstraction for the underlying hardware will open the door to plug and play solutions.

There are two key components in WARP that have to be enabled to dynamically react to changes in the antenna hardware in order to move the system to a real plug and play capability: Workflows and processors. In the following, we will discuss how this can be achieved.

B. Device Objects as Abstract Representation of Generalized Hardware

In order to enable the workflows and processors to adjust their programmed logic to changes in the underlying definition of L2 parameters, we chose an object oriented concept (see Figure 2). Every piece of hardware is represented by a class and all these classes are derived from a common base class. The first layer of inheritance specifies the type of device, e.g. up-converter, down-converter, high power amplifier and so on. Up to here, the classes are purely virtual and do not contain any implementation – they just specify the generalized interface. Only the next level of inheritance does contain the actual implementation. Therefore there is one class for each type of device in use at this level.

The concept of inheritance, precisely multiple inheritance in this case, allows to take advantage of another feature: Although being different in their type, devices from the same vendor most often have very similar interfaces. So in addition to being derived from a particular device class, let's say specifying a class to represent an up-converter, the same class can be derived from another base class representing the vendor. As shown in Figure 2, both up- and down-converter of the same vendor do derive common functionality from a vendor-specific base class. That way we can avoid duplicating code for similar devices of different type.

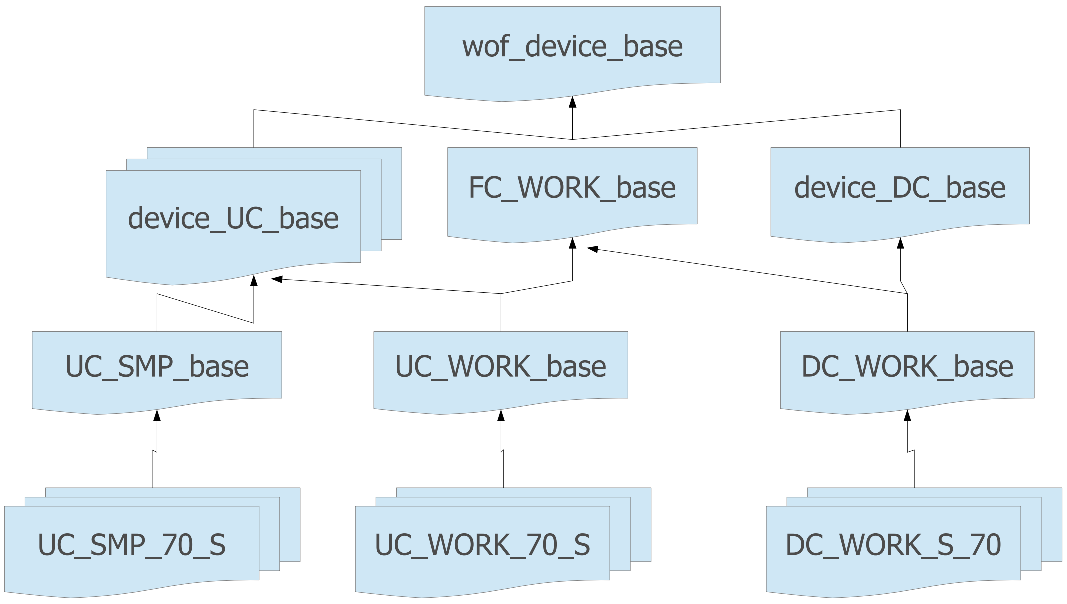

Figure 2: Class hierarchy for device-classes. As an example here limited to different types of S-band UC and DC devices of two different vendors.

In the same manner, we can further structure the inheritance for devices of same type and vendor, but with different specifications. To continue our example of up-converters, there are devices of the same vendor with the same functionality that just differ in the frequency-band they are targeted to work with. Most likely neither the parameters describing those devices, nor the protocol to communicate to them will differ, just the allowed range for the frequencies will. Again, this last layer of inheritance follows the programmers logic to not define the same thing twice at different places. From the systems point of view it is an unlikely option to replace an S-band up-converter by another one providing output in X-band frequencies. On the other hand, this demonstrates the power of the described concept – even reconfiguring the whole RF-equipment of an antenna to serve for a completely different frequency regime would be almost transparent from the M&C-systems point of view.

Figure 3 demonstrates the definition and implementation of the class methods.

- The base class wof_device_base only declares two very basic functions common to any device controlled by WARP: A method to configure the communication between generator and device – typically defining the IP settings address and port-number – and a generic function reset dedicated to bring any device in a well defined and save condition, regardless of its prior configuration.
- The next layer, wof_UC_base, declares all the functionality commonly provided by any up-converter in use: The output can be switched on and off and its frequency can be selected. This is the very point where the abstraction of a specific device takes place. The given hardware may have additional functionality, let's say it can perform a frequency-sweep whose shape may be configured with a manifold of additional parameters. But if this functionality would be needed in general, it'd have to be implemented in all devices of this type. If not, for instance because a frequency-sweep is performed by the base-band unit instead, we do not implement this feature to keep the devices compatible and exchangeable.
- The third level now specifies the vendor and by this the particular piece of hardware. At this level we know how to communicate to the device: which IP-port to connect to, which units to use for frequencies, the syntax of switching the output and so on. Here the so far virtually defined methods are implemented with respect to the particular hardware.
- Up to the third level, we do not even need to take care of parameter limits. If one tries to command a frequency out of range, there are many places down the command-chain able to take care of this. In WARP, this task is located at the generator. The generator (note: being aware of the particular type of hardware it is connected to) will forward only valid commands to the device and reject all others. But one may decide not to do such a check in the M&C at all but rely on the hardware if it executes or rejects a command.
  However, there are cases where our particular device class has to be aware of those limits: here the generic method reset. Since the intention of this method is, to bring the device into some well defined state, we have to explicitly choose a frequency at this point. And this frequency must differ depending on the

allowed frequency range.

Stressing the principle of inheritance, we can even define the parts of the reset method (e.g. switching off the carrier) in wof_UC_SMP_base and only the part setting the frequency in wof_UC_SMP_70_S.

Now as we have seen that an object oriented structure and inheritance allow to provide an object-representation for each device, the next question to address is how the workflows and processors are able to make use of those objects on run-time.

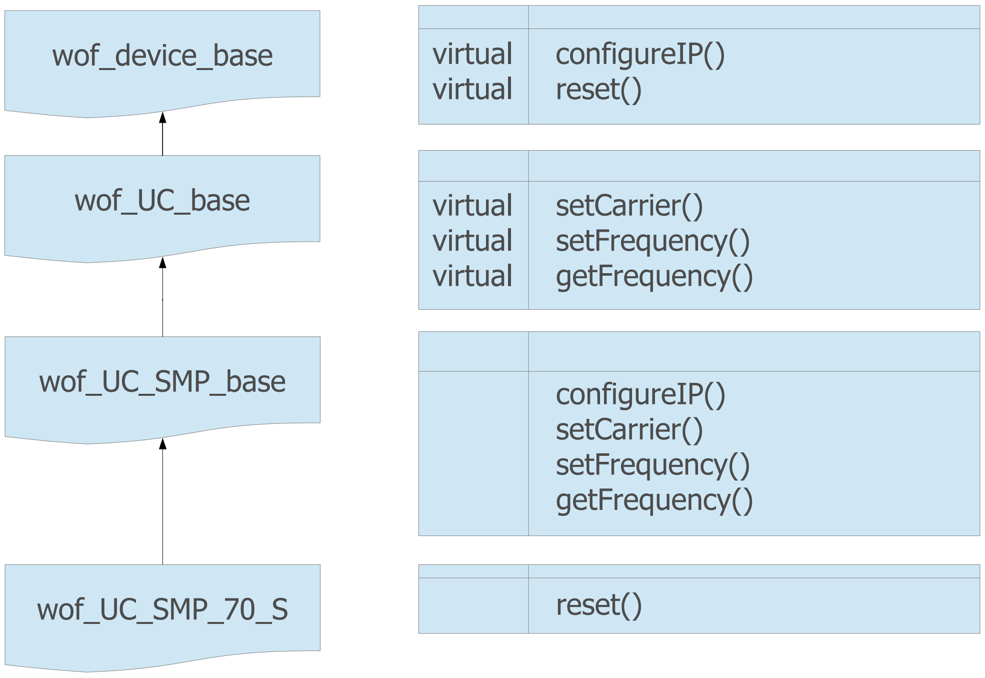

Figure 3: Virtual definition and concrete implementation of methods, here for example in case of a particular type of S-band up-convertor.

C. Coding Workflows in an Abstracted Manner

What we have completely ignored so far is, what the methods defined in the object model above actually do. We have shown that a device object has access to all information necessary to specify how to communicate with the hardware represented. But we did not specify how this communication actually happens – for good reason. One might be tempted to perform the actual commanding within the class methods, but that would unwillingly limit their functionality drastically.

To understand this, we have to step back from the low level implementation of the device-classes and widen the focus to the tasks to be fulfilled by the workflows, being supposed to make use of those classes. As discussed in much more detail elsewhere[2,3] the workflows in WARP are not only scripted command sequences to be blindly executed. The workflows have access to the complete monitoring of the controlled hardware. This monitoring on one hand defines on run-time the behavior of the workflow, on the other hand the monitoring serves as a feedback to judge, if the sent commands have been executed correctly. In addition, the workflows communicates with a state-machine representing the status of the antenna as a whole. Altogether this allows for generic workflows adapting themselves to special situations e.g. existing or missing redundancies and so on. Furthermore it allows the workflows to evaluate abnormal conditions and perform a sophisticated error-handling and even automated error-correction. It has been a great deal of effort to implement the workflows in the described manner, but it was the key to the biggest improvements of WARP over its predecessor.

Without going into details, it should be clear that it would be the wrong way to feed all these information from the workflow down to the device-classes. Much more promising is the other way around. The device-classes provide snippets of command sequences and/or checks on monitoring data, and the workflow is free to put those snippets together with all the logic needed. For example: Setting the frequency at an up-converter usually would go along with a verification in the monitoring for the right frequency to be set. Contrary, some trouble-shooting workflow intended to quickly react to some failure must not waste time with verifications based on cyclic monitoring. But both workflows shall use the same method provided by the same class. In other words, the device-classes provide the spelling, how to phrase commands in order to be understood by the particular hardware, but they do not talk themselves.

A nice side-effect of the usage of the device-classes is, that workflows themselves have become much more easy to maintain than before. In the previous version, the commanding was specified in terms of the particular L2-command-parameters, typically by a pair of parameter-name and -value, both being determined by the particular hardware. As an example, the command to switch the output of some frequency converter on and off is sometimes named “carrier”, sometimes “mute”. In order to keep the naming convention consistent with the device documentation (as system engineers are used to that), those names and their meaning have also been used as parameter names within WARP. In combination of several pieces of hardware that lead to the odd situation that a command sequence had to use carrier=1 and mute=0 for different devices in order to reach the same result. Even worse, the person writing the workflow had to be aware of this difference and hat to keep track on where to use which notation.

Now, having the functionality abstracted within the device-classes, the same command reads the same for all devices, regardless the name and value of the underlying parameter. And a system engineer working on L2-parameters underneath the layer of abstraction, still sees her familiar parameter names.

D. Changing Hardware at Run-Time

So far, we have discussed how the desired task description of a workflow can utilize hardware abstraction to become independent of the particular hardware in use. But the meaning of plug and play also includes that changes in the hardware can be done at any time with the system reacting to these changes at once.

A more technical formulation of the same question is: How does the workflow decide on run-time, which type of device class to instantiate? A solution truly called “plug and play” would require the system to realize the changed type of hardware automatically. As we have seen, this is impossible within WARP as at least we have to exchange the generator, the application providing the interface between WARP and the hardware. The next to best solution is, there is just one single adjustment needed to reconfigure the whole systems, once the generator has been replaced. It is this solution we are aiming for.

Aside the processor within WARP, we have included another application, the resource manager RMG, to provide information on the availability and type of all connected components. Originally, the RMG was integrated to have a mechanism to mask out devices disassembled from the antenna for a limited time. In the context of hardware exchange, it is the perfect place to allow further to specify the particular characteristics of the devices in use.

In terms of hierarchy, the RMG is something in between L2-generators and L3-processors. Like generators, the RMG provides and propagates information, allowing all processors GUI's and other applications to make use of this information. From a more logical point of view, it resides above generators and their L2-parameters, as it controls in how far generators are able to provide their data at all. However, as already pointed out, the distinction between L2 and L3 is more a logical than a technical one, and the centralized PDB-server assures that all necessary components receive the information provided by the RMG.

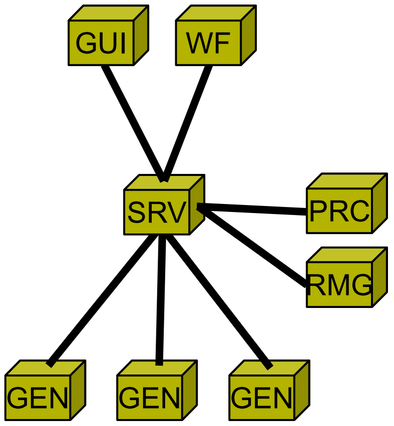

Figure 4: Schematic structure of WARP including a resource-manager RMG.

More important than the classification as being L2 or L3 is the fact, that the RMG is part of the real-time M&C-system, not an external configuration managed off-line. So a change in the RMG-settings will be propagated to the entire system the very same way, a change of any device parameter will be propagated. That way, all components will always access the actual settings of the RMG-parameters at any time.

For a workflow, triggered at some point to be executed, this means it first retrieves the information on the available hardware from the RMG, then instantiates all device-classes according to this information and finally generates the required command sequences to be executed. As workflows being executed at certain points in time and do not run permanently, the instantiation of the device-classes will happen for each workflow at run-time providing for each workflow the correct environment.

E. Coding Processors in an Abstracted Manner

Besides workflows, the second essential point to make WARP ready for plug and play is the processor. But most of the development for workflows can be easily translated to processors as well. Both applications access the same data in the same way – provided as parameters by the PDB-server. Actually, both applications have their logic written in the same language, since both workflows and processors use LUA-scripts[4] to code their functionality. The only difference between the two – and that being a severe one – is, workflows are externally triggered to run once for a short period of time, while the processor script is started once and then runs continuously. Therefore the concept applied to workflows, instantiating the device-classes at start-up, is not applicable to processor scripts.

However, the integration of the resource manager into WARP provides an elegant solution. To work with real-time monitoring data, any processor works reactive on data sent by other components of WARP. Now, the RMG is nothing else than such a component and any change in the configuration provided by the RMG will be sent to the processor the same way. Just the call-back for RMG-parameters has to look different to other parameters; it has to delete the previously instantiated device-classes and create the new ones taking into account the changed hardware.

With this functionality, one can start to code the logic to derive L3- and L4-parameter from the basic L2-information provided by the generators using functionality provided by the device-classes the same way, the workflows do create their command sequences.

F. Self-Identifying Hardware

As pointed out, creating a system truly realizing plug and play solutions, shall work without an external interface like the RMG, that depends on separate inputs from outside, announcing changes of the hardware setup. However, a real plug and play mechanism relies on the cooperation of the plugged hardware to identify itself. For antenna equipment, this is at the moment far beyond imagination. Even Ethernet-interfaces, with no doubt being state of the art by now, are not incorporated to all antenna hardware brought to the market, not to speak of protocols. Therefore, for the near future, there is no hope that there will be a common standard, that identifies any hardware in order to allow the connected M&C to adapt its communication to the specific hardware.

Nevertheless, one can try to realize such a scenario within the M&C-system as much extended to the external interfaces as possible. In our case, the generator would be the one identifying itself together with the specific device it is designed to communicate to. Technically, the design of WARP allows to realize such a scenario. The RMG is implemented as a processor, therefore aware of L2-parameters provided by the generators. Therefore in principle, any generator could identify itself on start-up, send this information to the resource-manager, and the RMG in return could provide this information to workflows and the L3/L4-processor. However, at the moment this is only a conceptual analysis but nothing planned to be realized with high priority.

Another thing is worth to be mentioned here: When setting up the hierarchy for the device-classes it was realized that certain devices of the same vendor have similar protocols implemented. In consequence we have unified the WARP generator for those devices. As requesting monitoring information from all of these devices is done in the same way, we can evaluate the response that contains information on the device model and version. This way we were able to create a generic generator for a variety of frequency-converters. With the first received monitoring block the generator adjusts itself to the particular type of device and is further on able to interpret the following monitoring blocks in a correct way. Again, for now this applies only to a limited set of devices provided by the same vendor. But it is a proof of principle that plug and play solutions can be realized even for as specialized hardware as antenna RF-equipment.

III. Plug And Play in the Context of CCSDS Service Management

In the previous section we have discussed plug and play solutions under the perspective to improve the efficient use of ground station equipment. The same developments are also of interest, if seen in the broader scope of a missions complete ground segment. Gaining efficiency here means, to enable the satellite operator to coordinate the various parts of the ground segment in a most easy and potentially automated way. Within the Consultative Committee for Space Data Systems, the “Cross Support Services” working area develops standards, used for communication between the satellite operator (mission) and the ground segment provider, here seen as a service provider. In a rough summary, there are three things to describe:

1. A service provider – in our case the ground station – has to advertise, which services can be performed at all utilizing its hardware. This will be called a “Service Catalog”.
2. Between the service provider and the user, it has to be negotiated, which of those services will be requested and which parametrizations are to be applied. In CCSDS terminology this is a “Service Agreement” and the parameterizations are called “Configuration Profiles”.
3. On a well defined interface, the user has to have the ability to request certain services with specified settings for certain times. This is called “Service Planning” and “Service Requests”.

The first two of those three points are the ones of interest in the context of this paper. For the service offering, the ground station has to describe what can be done with its equipment, without going into details concerning the technical implementation at the ground station. In turn, the user does not care about the particular hardware utilized at the ground station to carry out some task. Especially the mentioned parametrizations have to be formulated in an abstract way, not using the parameter language of the underlying devices. In CCSDS's terminology, the abstracted view on the hardware is called “functional resources”. This is the same concept that also was the design driver for WARP and the workflow description within WARP. However, as we have discussed in the previous section, a plug and play solution requires more than just abstraction. And the two major decisions to make are, where in a system the translation between device dependent parameter description and abstracted task formulation is implemented best and to which extend the available functionality is covered by the abstract formulation.

It turns out these decisions have an interesting parallel to a decision needed to implement CCSDS service management. While CCSDS defines a language to specify tasks and an interface to request them, it is not part of the standardization how generic or specific those tasks are. For very good reasons it is up to each service provider to decide, how many details of the existing hardware are exposed to the user. If found to be useful, the ground station could define its services in such a fine granularity that basically each and every command possibly sent to the antenna hardware corresponds to an own service or an own parameter describing such a service. That way, the antenna could be completely controlled by the external customer – but for sure, that is nothing a ground station is going to accept. Instead, the ground station will carefully analyze, to which extend it allows to steer functionality from the outside. Obviously the service provider will keep the user as far away from low level commanding as needed to assure the user can not do any harm to the station hardware.

Even if CCSDS does not define a mandatory set of services and parameters to be used for service management, it gives examples how typical services may look like. Such an example, taken from the CCSDS documentation[5], for functional resources needed to describe AOS Forward and Return services is shown in Figure 5. It illustrates that the building blocks needed to define the services are in terms of detail somewhere in between the level of hardware devices and the level of single commands sent to the hardware.

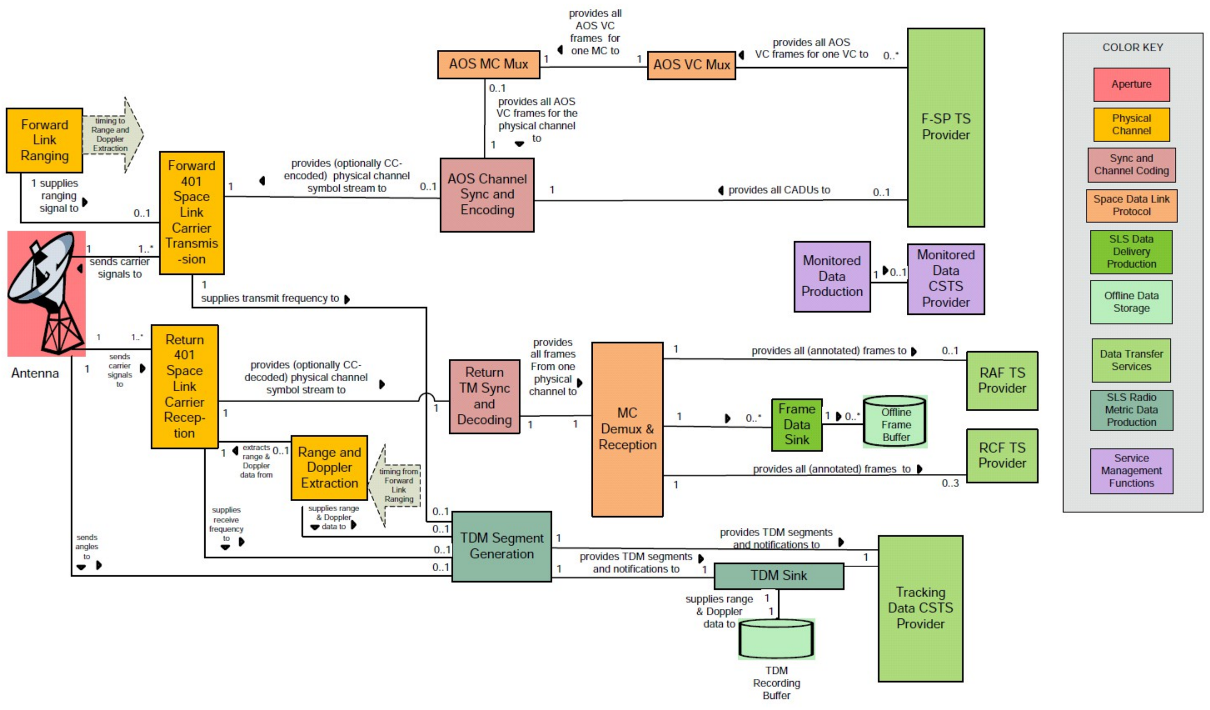
provides all AOS VC frames for one MC to
AOS MC Mux
AOS VC Mux
provides all AOS VC frames for one VC to
F-SP TS Provider
COLOR KEY
Aperture
Physical Channel
Sync and Channel Coding
Space Data Link Protocol
SLS Data Delivery Production
Offline Data Storage
Data Transfer Services
SLS Radio Metric Data Production
Service Management Functions
provides all AOS VC frames for the physical channel to
Forward Link Ranging
timing to Range and Doppler Extraction
Forward 401 Space Link Carrier Transmis-sion
provides (optionally CC-encoded) physical channel symbol stream to
AOS Channel Sync and Encoding
provides all CADUs to
supplies ranging signal to
sends carrier signals to
supplies transmit frequency to
Monitored Data Production
Monitored Data CSTS Provider
Antenna
sends carrier signals to
Return 401 Space Link Carrier Recep-tion
provides (optionally CC-decoded) physical channel symbol stream to
Return TM Sync and Decoding
provides all frames From one physical channel to
MC Demux & Reception
provides all (annotated) frames to
RAF TS Provider
Frame Data Sink
Offline Frame Buffer
RCF TS Provider
provides all (annotated) frames to
extracts range & Doppler data from
Range and Doppler Extraction
timing from Forward Link Ranging
supplies receive frequency to
supplies range & Doppler data to
sends angles to
TDM Segment Generation
provides TDM segments and notifications to
provides TDM segments and notifications to
TDM Sink
Tracking Data CSTS Provider
supplies range & Doppler data to
TDM Recording Buffer

Figure 5: Functional Resources for an AOS Mission Service Agreement as defined by CCSDS[5]

How does this help towards our goal to implement plug and play solutions at ground stations? The above is basically the appropriate scheme to define the interfaces of our device classes. We can deduce from the CCSDS service management description, which parametrization is necessary but sufficient to describe relevant services. Anything added to this is not required by the service, but corresponds to specifics of the particular hardware. Such add-ones will depend on the particular type of the device and hence must not be part of the interface of the device class. In other words, the functions and parameters existing in the service definition have to map to methods of the device classes, everything else must be hidden by the interfaces of the device-classes and just shall determine the concrete implementations.

IV. Plug And Play for Ground Network Components

Another system based on the SpACE-framework is NEMO[3], a monitoring and control system for the entire IT-infrastructure at GSOC. Using the same framework, the architecture of NEMO equals the one of WARP as discussed above and depicted in Figure 1. By now, NEMO provides monitoring of more than 200 servers, physical hosts as well as virtual machines, and of about 150 network elements, such as switches, routers, firewalls and so on. In addition, NEMO also allows to control these components.

The content of the monitoring, especially collected at the servers, can be grouped in two different parts: Basic data from the server itself – typical examples being CPU-load, memory consumption, disc-space and so on – an data from the applications running on those servers. Concerning the information on applications, again one can distinguish between data on process level provided by the host's operating system and data specifically provided by the application itself through some dedicated interface.

The same classification is appropriate to group commands. There are commands to the host itself, e.g. bring up/down a virtual network-interface; system commands to start/stop applications; and finally commands sent to some running application.

In the context of plug and play solutions, we have to distinguish between data exchange with the operating system of an host and communication with some application. If a server is replaced, we can assume that the new one has either the same operating system (OS) installed as the previous one, or the OS is at least of the same type, e.g. some Linux distribution. In both cases, the way to request information from the OS or issue commands to the OS will stay the same. So we can use the same generator than before to interface between NEMO an the OS. And within NEMO, the same parameters are distributed by the PDB-server to the other components of NEMO. In other words, in case of IT-infrastructure, the hardware (or virtual hardware) is already standardized in a way that allows to exchange it with least effort.

In case an application is replaced by another version or a different one with equivalent functionality, the situation is more similar to what we have discussed for antenna equipment. Interfaces to the applications are not standardized and may even differ from version to version. Now we can make use of the same developments already achieved in the context of the antenna-M&C WARP. As WARP and NEMO have the same architecture and their components are parts of the same framework, we have the same two places – workflows and processor – to place a layer of abstraction, technically implemented by the so called “device-classes”.

It is worth to note that also the work done in the “Cross Support Services” area of CCSDS blurs the difference between antenna hardware and ground segment software. As can be seen in Figure 5, the defined service includes hardware components as well as software applications (e.g. data transfer services color-coded in green). Within the service management both parts are treated equally. Having a common framework for M&C-tasks for both, ground-station equipment and ground-segment network elements, we can line up to the service oriented approach as proposed by CCSDS. In other words, the control-systems in use at GSOC are perfectly suited to allow operations in the way visioned by the “Cross Support Services” area of CCSDS.

V. Conclusion

In modern IT-systems, Plug and Play solutions have become common. We are used to just plug keyboards, printers, storage devices, cameras and whatsoever into some USB-port and use the freshly connected hardware with no need to install dedicated interface drivers. For equipment integrated at a ground station, such an easy integration is up to now far from realistic. However, although all the devices do communicate only through proprietary protocols and hence require dedicated interfaces, we can implement plug and play solutions within the connected monitoring- and control-system to some extend.

The basis for such solutions is an abstracted definition of services or tasks to be fulfilled by the hardware in question. This functionality can be used to define the interface of so called device-classes, hiding the actual implementation of some particular piece of hardware within the particular instantiation of the device-classes. With the concept of inheritance, these device-classes can provide the translation between the device specific parametrization and an abstracted task: the task specifies the abstracted interface of base-classes, that covers the required functionality, while the particular implementations take care on how the tasks are to be carried out by a certain piece of hardware.

At Weilheim ground station, we have successfully defined and implemented such device-classes at first for up-converters. Using these classes, we were able to make all our operational workflows independent on the particular device types integrated to the various antennas. A first use-case was the replacement of up-converters in one of our multi-mission S-band antennas. We were able to continue operations with the given workflows, without changing them in any way. The system was adapted to the altered hardware setup by changing the selected model of the devices within the resource-manager. Following this successful proof of principle, we are currently determining the necessary interfaces to create device-classes for other types of equipment. For doing this, we analyze the content of our operational workflows, as well as the examples given for service descriptions in the context of the "Cross Support Services" area of CCSDS. Finally we plan to also formulate the logic of the processor of WARP in terms of the abstracted methods provided by the device-classes.

Widening the scope from ground station hardware to the whole ground segment, we have shown that plug and play solutions and service oriented approaches are perfectly in line. To work within service oriented approaches requires to have abstracted the implementations. In turn, this abstraction allows for plug and play. The monitoring- and control-systems in use at GSOC are well suited to allow both.

Appendix A
Acronym List

AOS	Advanced Orbiting Systems – Space Data Link Protocol
CCSDS	Consultative Committee for Space Data Systems
DC	Down-Converter, Frequency-converter from RF to base-band frequency
DLR	German Aerospace Center
GEO	Geostationary Orbit
GSOC	German Space Operations Center
GUI	Graphical User Interface
HPA	High Power Amplifier
IOT	In-Orbit Testing
IP	Internet Protocol
LEO	Low Earth Orbit
LEOP	Launch and Early Orbit Phase
LUA	A powerful, fast. Lightweight, embeddable scripting language
M&C	Monitoring- and Control-System
NEMO	Network-Monitoring, the new IT-infrastructure M&C at GSOC
OS	Operating System
PDB	Parameter-Database, internal data exchange protocol used by our M&C-systems
PRC	Processor, application within WARP to calculate derived information
R&D	Research and Development
RF	Radio Frequency
RMG	Resource-manager, application within WARP to control the availability of hardware
TT&C	Telemetry, Tracking and Command
UC	Up-Converter, Frequency-converter from base-band frequency to RF
USB	Universal Serial Bus
WARP	Weilheim Antenna Remote Processing, Weilheim's new M&C-system
WF	Workflow, scripts to execute predefined tasks, ans also Workflow-GUI, application within WARP to interactively execute workflows

Acknowledgments

The Authors would like to thank all members of the software group at GSOC's department "Communication and Ground Station", who all contributed significantly to WARP and NEMO: Christopher Buschor, Rossella Falcone, Sylvain Gully, Thomas Ohmüller, Stefan Veit, Yi Wasser and Daniel Weber. Furthermore we wish to thank all the colleagues at GSOC from the various departments, who have contributed with their particular expertize, especially all the system engineers at Weilheim ground station. Finally we want to express our thanks to all the shift-operators at GSOC and Weilheim. Their feedback based on the experiences made within the real operations-environment was invaluable.

References

[1] D. Dikanski, M. Preuß and K. Wiedemann, *"Dual Operation of TerraSAR-X and TanDEM-X with One Ground Antenna"*, presented at SpaceOps 2012.

[2] A. Hauke, T. Ohmüller and U. Häring, *"An Innovative Monitoring- and Control-System at GSOC and Weilheim Ground Station"*, presented at the Ground System Architectures Workshop, GSAW, 2012. *http://csse.usc.edu/GSAW/gsaw2012/s1/hauke.pdf*

A. Hauke and E. Barkasz, *"Multi-Mission Support with WARP"*, presented at SpaceOps 2012.

[3] A. Hauke, E. Barkasz, U. Häring and M. Preuß, *"Reality Filtering"*, presented at SpaceOps 2014.

[4] *https://www.lua.org*

[5] The Consultative Committee for Space Data Systems, CCSDS 902.0-G-1, *"Extensible Space Communication Cross Support – Service Management – Concept"*, 2014.