ARM instrumentation of the PHOENIX ATC System for Performance evaluation

Dr. Kai Engels, MyARM GbR; and DFS SH/T
Ralf Heidger, DFS Systemhaus (SH), Head of SH/T
Stefan Ruppert, MyARM GbR

Dr. Kai Engels, Stefan Ruppert
MyARM GbR
Neue Str. 4; 63571 Gelnhausen-Roth; Germany
Phone: +49 6192/9772818
Fax +49 6192/9772819
web: http://www.myarm.de
email: info@myarm.de

Ralf Heidger
DFS Deutsche Flugsicherung GmbH
Systemhaus; SH/T
Am DFS Campus 7; 63225 Langen; Germany
Phone: +49 6103/707-2520
Fax: +49 6103/707-2595
email: ralf.heidger@dfs.de

1. Introduction

Air Traffic Control (ATC) systems are highly distributed systems. Nationwide distributed radars provide measurement data through digital wide area networks (WAN) like the radar data network (RADNET) from an area under control (e.g. Germany). This radar data runs through a sequence of steps in the radar data processing systems (RDPS) on the air navigation service providers (ANSP) side, before it is displayed on a so-called controller working position (CWP). These processing steps are executed by many different processes in an Air Traffic Control (ATC) system on various computers connected via a local area network (LAN) or a wide area network. This distributed nature of ATC systems makes it difficult to get an insight view of these manifold processing steps and their performance as well as the throughput of radar data through the whole chain, thus being incompletely aware of all possible radar data processing bottlenecks.

To improve the quality control technology of radar data processing the DFS instrumented the core functions of its operational radar data processing system PHOENIX [Euler, B.; Heidger, R. (2007), Heidger, R. (2006)] to get an insight view of the performance of the distributed processing chain. PHOENIX is the operational RDPS at all national and international airports in Germany with more than 100 deployed CWPs. Thus the system's performance is directly linked with the German air traffic management efficiency, which is of vital interest for the DFS. The source code instrumentation for that analysis was implemented with the Application Response Measurement (ARM) standard version 4.0 [The Open Group (2004a)] from The Open Group and two ARM implementations, among them the MyARM [Ruppert S. , Engels, K. MyARM GbR] as the ARM measurement and analysis tool.

2. The MyARM toolkit at a glance

The MyARM agent implementation provides full compatibility to the ARM 4.0 standard for C and Java language bindings. With version 1.1.0 the Apache Portable Runtine (APR) library is used as the OS abstraction layer so MyARM will support

almost any UNIX®, Microsoft Windows® and MacOS X operating systems. It has a modular backend/frontend concept supporting various kinds of efficient data transport, i.e. shared memory or TCP/IP and storage mechanisms for the measured performance data, i.e.

- MySQL and SQLite and on demand any database which is supported by the APR database abstraction layer such as Oracle or Postgres database.
- XML files for ex- and import of measured ARM data.

MyARM provides three different kinds of analysis tools:

1. Command line tools for batch and mass data processing of ARM data.
2. Web-based analysis tools by using standard web-browsers like Mozilla Firefox or others to support remote and quick analysis of measured transactions.
3. Manager graphical user interface (Image 1) based on the Qt® 4 C++ class framework from the MyARM cooperation partner Trolltech to analyse and manage measured ARM data.

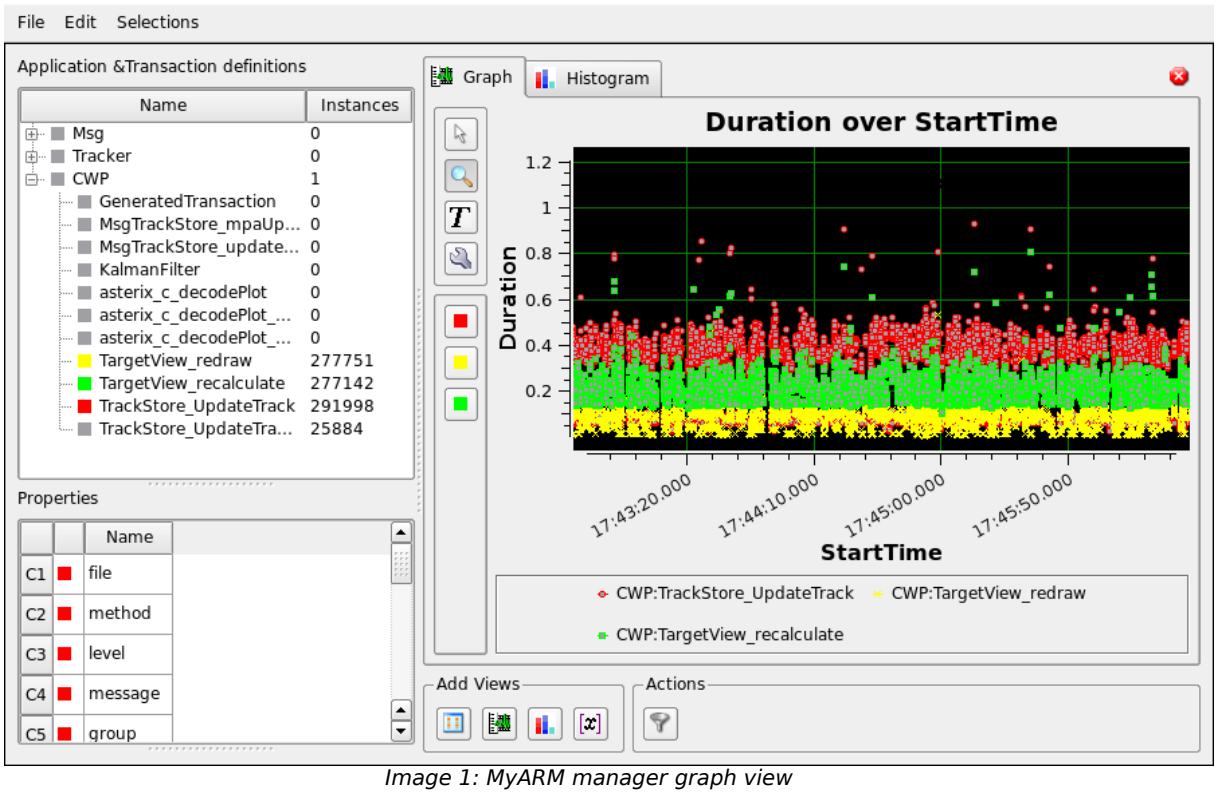


Image 1: MyARM manager graph view

Besides the ARM agent and analysis tools MyARM provides ARM 4.x instrumentation support. With the QArm framework ARM 4.x instrumentation can easily be integrated into any Qt® 4 based application. Moreover, instrumented versions of the popular SQL databases SQLite and MySQL can be downloaded from the MyARM web-site and last but not least our special and much improved version of the Apache 2.x mod_arm4 module is provided.

3. PHOENIX System architecture

PHOENIX is an efficient, distributed, client-server-based multi-radar tracking and air-situation display system, running on the Linux operating system. The PHOENIX multi-radar track server (MRTS) tracks radar measurement data by Kalman Filtering and generates aircraft tracks, which are representing the statistical optimal estimation of the state of a aircraft flight. A message server correlates flight plan data to such tracks enabling controllers to get information about a flight on the

radar screen. 3 – 5 (at an airport tower) or up to 120 (in an area control center) controller working positions (CWP), which are running on specific client computers. The following diagram (Image 2) shows an example network configuration with one track server, one message server and three controller working positions running on different hosts in a network.

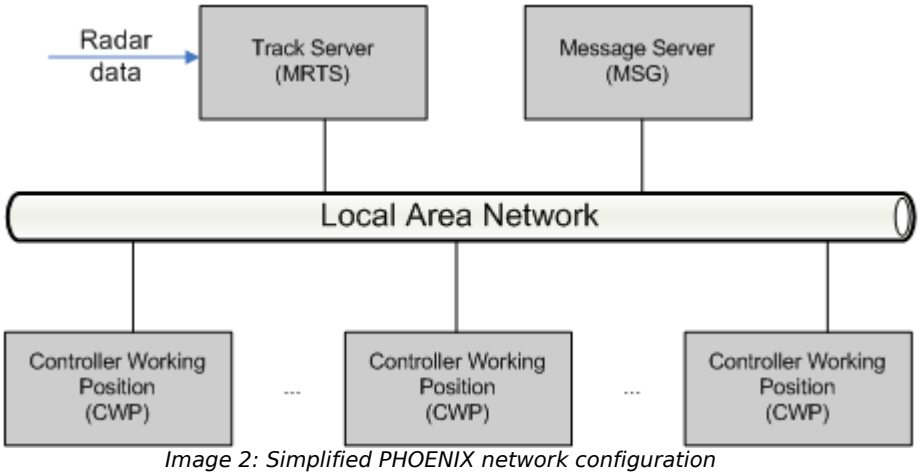


Image 2: Simplified PHOENIX network configuration

PHOENIX is far more complex than described above with more than 30 processes involved. The processes we concentrate on in this evaluation provide the core features of PHOENIX, therefore these processes have been instrumented with ARM. The instrumentation points are located in the processing steps depicted in Image 3, providing a measurement of the central processing points in the main data flow chain. Image 3 also shows the data flow as described above with a message server as relay between the tracker and the CWP. Only one CWP is shown in this image but, as depicted in Image 2, it can be more than one.

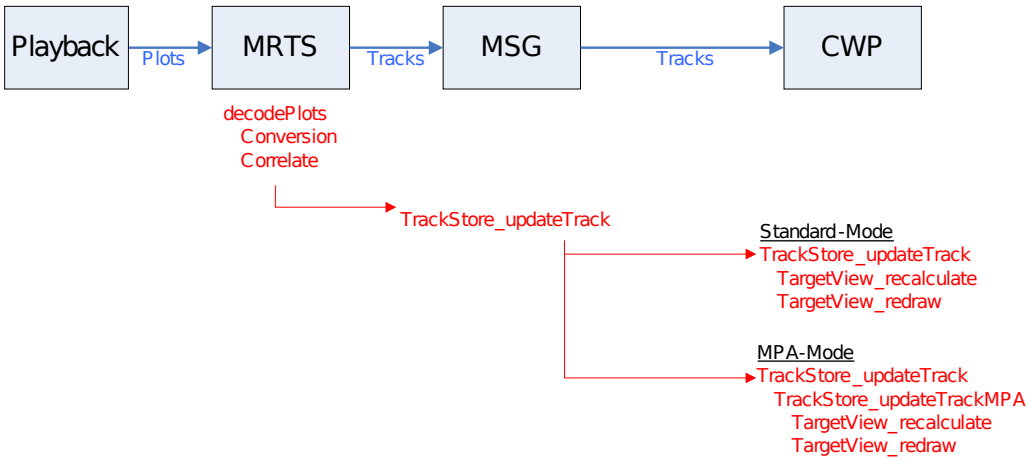


Image 3: Data flow and instrumented processing steps of the PHOENIX configuration used for the ARM evaluation

The plot to track processing inside MRTS includes the following processing steps:

- *decodePlots* is the parent transaction of all following processing steps, i.e. each *redraw* transaction in the CWP has exactly one *decodePlot* Transaction as parent. *DecodePlots* is divided into two sub transactions: *Conversion* and *Correlate*.
- *Conversion* is the transaction that converts the received ASTERIX plots into an internal format which can then be further processed.
- *Correlate* is the transaction doing the main work, i.e. either correlating received plots to an existing track or creating a new track from a plot that

cannot be correlated. Moreover, a decision is taken here whether a plot leads to a track update message or not.

The MSG has only one main processing step:

- *TrackStore_updateTrack* is the main processing loop of the MSG correlating flight plan information to each received track update and forwarding it further to the CWP.

The following steps take place in the controller working position:

- *TrackStore_updateTrack* is the same method using in MSG server since both rely on the same source code. In the CWP this is the main transaction calling *TargetView_recalculate* and *TargetView_redraw* for each received track update under normal circumstances (Standard-Mode).
- *TrackStore_updateTrackMPA*: In case the CWP works under heavy load an overload prevention algorithm is activated that collects all received track updates and calls *TargetView_recalculate* and *TargetView_redraw* only every 500 milliseconds (Multiplot-Averaging (MPA) Mode). This mechanism was introduced to avoid performance problems during high traffic load but until the ARM evaluation its efficiency could not yet be verified exactly.
- *TargetView_recalculate* recalculates the position of a received track update and prepares it for redrawing.
- *TargetView_redraw* redraws the target on the screen.

Each processing step is modelled as a transaction, i.e. the duration of each step is measured individually. A full transaction chain starts with the *decodePlot* transaction of MRTS and ends with the *TargetView_redraw* transaction of the CWP.

4. Scenarios und test environment

The performance of PHOENIX was investigated with two types of measurement scenarios. The first scenario type consists of recorded live data of the whole German air space, lasting about 30 minutes from 14:01:17 to 14:28.58. At that time around 700 tracks had to be handled by PHOENIX. Image 4 shows the PHOENIX CWP presenting this scenario.

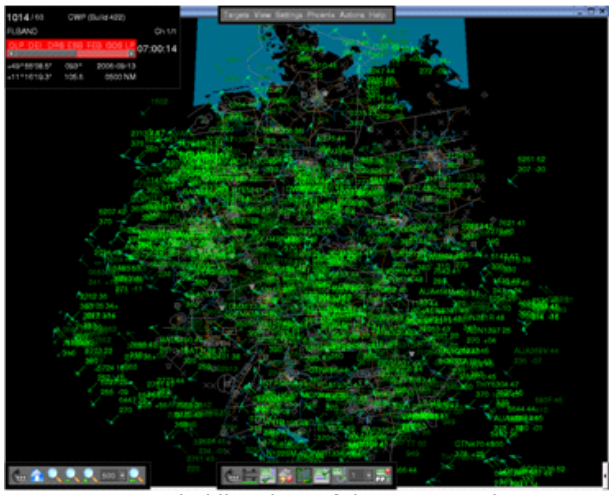


Image 4: Recorded live data of the German airspace

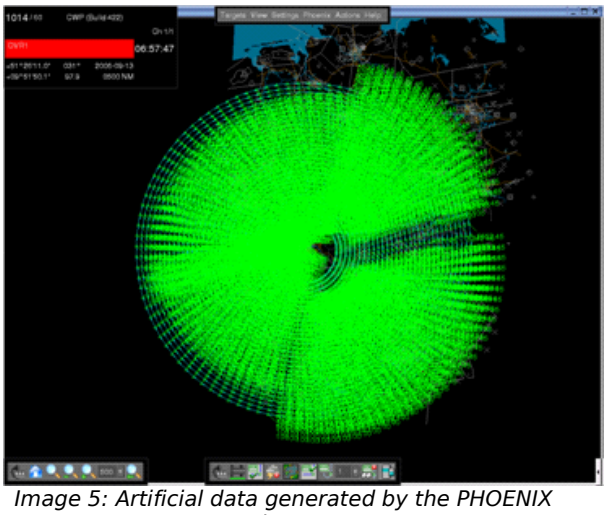


Image 5: Artificial data generated by the PHOENIX d-gen

The second scenario type is designed for high load analysis and consists of artificial data with 3000 tracks flying in 30 circles, 100 flights each (see Image 13). Average german traffic load varies between 500 and 1200 tracks in total during daytime, with peaks to 2000 and drops to less than 100 during night time.

Two different test beds were used during the evaluation. First, a standalone test bed with all three instrumented processes running on one host communicating over the local loop back interface. This test bed was used to get impressions on performance bottlenecks of the applications. Moreover, besides the single host environment, the instrumented applications were also executed in a distributed environment with one track server and 120 CWP computers to get measurement values in a realistic environment. The measured performance data of each run was stored in a MySQL database. After a scenario run the analysis of the data was conducted with the MyARM manager.

5. Synchronous vs. asynchronous duration

The ARM 4.0 standard is designed for the measurement of a synchronous duration, i.e. each parent transaction in a chain stops not till then when its child transactions have finished since it waits for the stop call of the child transaction (see Image 6 for an example).

In that standard case the duration of the parent transaction Δt_1 "includes" the duration Δt_2 and Δt_3 of its children. When looking at the whole chain it is enough to put the duration of each transaction in relation to the main duration to get an idea where a high delay might come from.

It becomes more difficult when asynchronous relationships exist, i.e. the parent transaction does not wait until the children have finished but stop the duration measurement asynchronously. In that case two possibilities exist on how the asynchronous duration of the transaction chain can be computed.

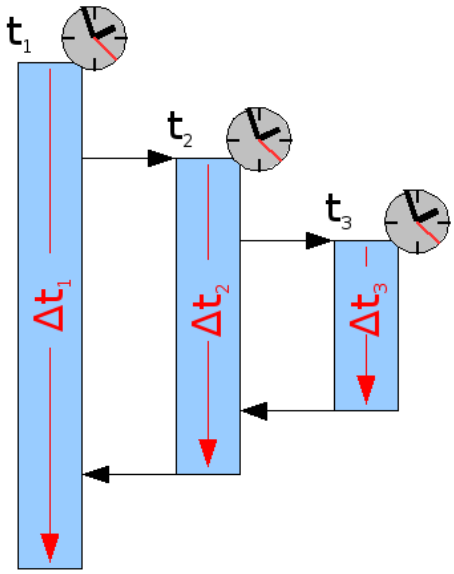


Image 6: Synchronous transaction duration

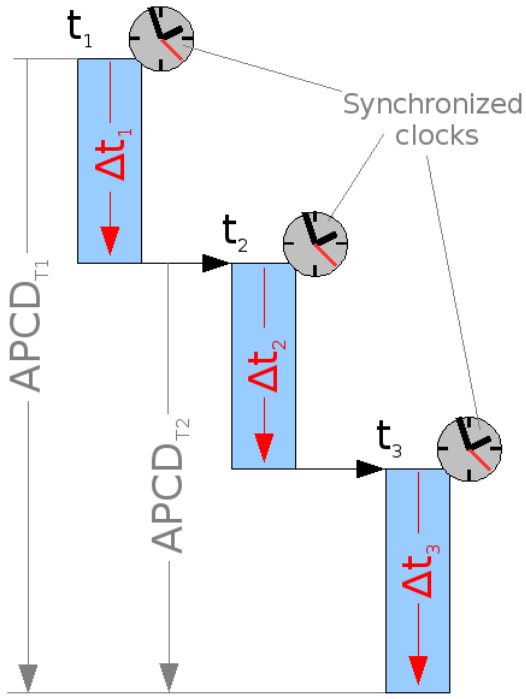


Image 7: Asynchronous parent child transaction duration

The first possibility is to compute the asynchronous duration of the transaction chain by traversing the chain from the root transaction (i.e. the transaction for which we want to know the asynchronous duration for) to the last child of the chain. The "last child" in that context means the leaf in the transaction chain with the highest start time. The asynchronous duration is then computed as the time difference between the stop time of this last child and the start time of our root transaction. This duration is referred to as "asynchronous parent/child duration", or APCD.

Let us have a look at Image 7 as an example. If we want to know the APCD of this transaction chain we compute it as follows. First we look at T_3 which is the leaf of the chain with start time t_3. The APCD of T_3 itself is ΔT_3 since T_3 has no

children. Going up one level we compute the APCD of T_2 as the time difference between t3 plusΔt_3 and t_2.

$$APCD_{T2} = (t_3 + \Delta t_3) - t_2$$

The APCD of T_1 is then the time difference between t_1 and t_2 plus the APCD of T_2.

$$APCD_{T1} = (t_3 + \Delta t_3) - t_1$$

The APCD can be understood as duration of the whole (sub-)chain.

The second possibility is to compute the chain duration starting from the leaves of a chain (which might be a tree). This is especially useful in cases where more than one leaf exists in a chain as depicted in Image 8. By traversing the tree from the leaf one transaction tree will produce as many asynchronous durations as there are leaves. The asynchronous duration computed that way is referred to as “asynchronous child parent duration” or ACPD. Image 8 shows exemplary how the ACPD is computed. The first duration value starts from transaction T_4, adding the duration of T_4 to the difference between t_1 and t_4 (i.e. taking the start time of the transactions into account and not the bare duration as it is the case for synchronous duration computation). The whole ACPD for T_4 is then computed as

$$ACPD_{T4} = (t_4 - t_1) + \Delta t_4$$

The second ACPD for T_3 is computed the same way, i.e.

$$ACPD_{T3} = (t_3 - t_1) + \Delta t_3$$

The different levels of the ACPD can be computed by choosing the name of the transaction leaf and the level on which this leaf must exist. To get the $ACPD_{T2/T4}$ the user would have chosen the transaction T_2 as root transaction and T_4 as last child on level 0 (relative to T_2).

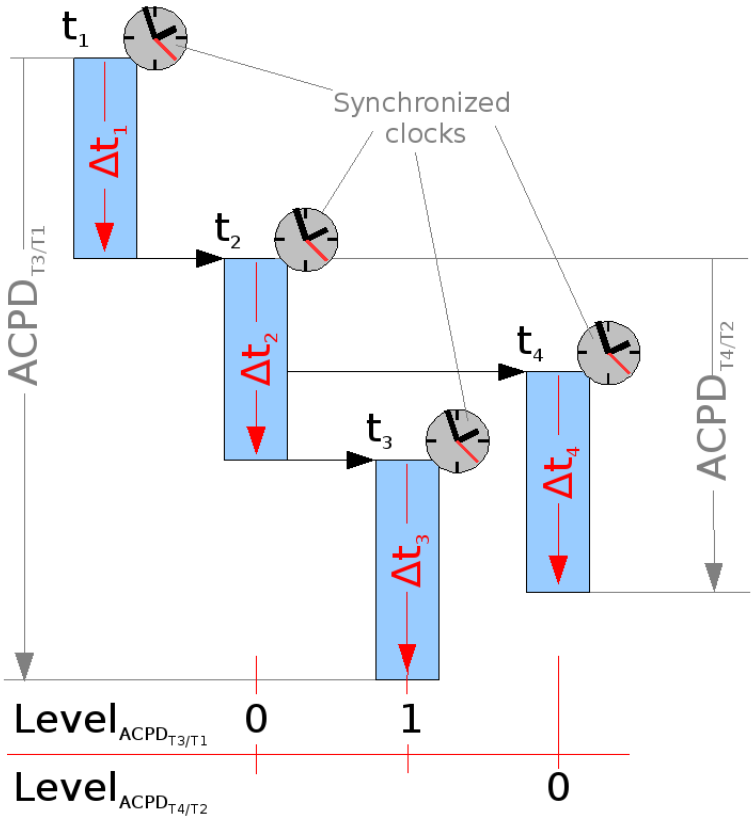


Image 8: Asynchronous child parent transaction duration

Both asynchronous durations should be used in cases

- where transactions are asynchronously stopped, i.e. do not wait until the child transactions have finished
- where all clocks that are involved in the measurement are synchronised (otherwise the computation will deliver complete non sense data since the start time is used as reference for the computation). This has to be assured especially in distributed environments with the network time protocol (NTP) for instance.

Neither APCD nor ACPD values are covered by the ARM 4.0 standard, i.e. these asynchronous durations where invented by MyARM to provide practical solutions in distributed environments where the synchronous duration does not give any hint of what is going on in the system.

6. ARM 4.1 and asynchronous durations

The ARM 4.1 [The Open Group (2007)] standard has been available since mid of 2007 and it covers the need of asynchronously executing transactions. At the time the PHOENIX performance evaluation was made the ARM 4.1 standard had not been completed. The list below gives a brief overview of improvements of ARM 4.1 which can be useful in the PHOENIX environment for future measurements:

- Correlators can be marked as “asynchronous” to indicate the correlation between two transactions is asynchronous in nature (e.g. no typical client/server transaction). Analysis tools can then correctly differentiate between synchronous and asynchronous transaction correlation thus it is now possible to have a mixture of such transaction relationships.
- Correlators can be marked as independent to indicate that the status and duration of the child transaction does not influence the status and duration of the parent transaction. In the PHOENIX environment it does not make sense to mark a transaction executed within the tracker to be FAILED if a transaction failed in the CWP.
- A the new instrumentation control interface of ARM 4.1 is available to optimize the ARM agent to reduce the ARM processing to a minimum within the instrumented application (if the ARM agent supports it).

Other improvements of the ARM 4.1 standard do not greatly matter in the PHOENIX environment and are therefore out of scope in this environment.

7. Duration of processing chains

A main requirement at DFS is that the delay between the reception of radar data and the presentation of that data to the controller should not exceed one second.

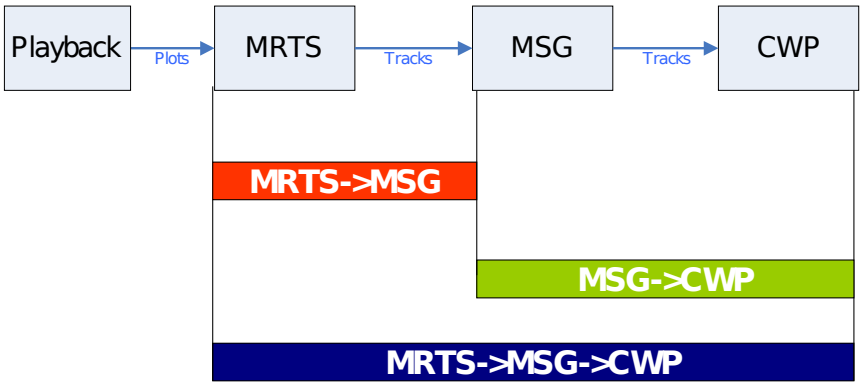


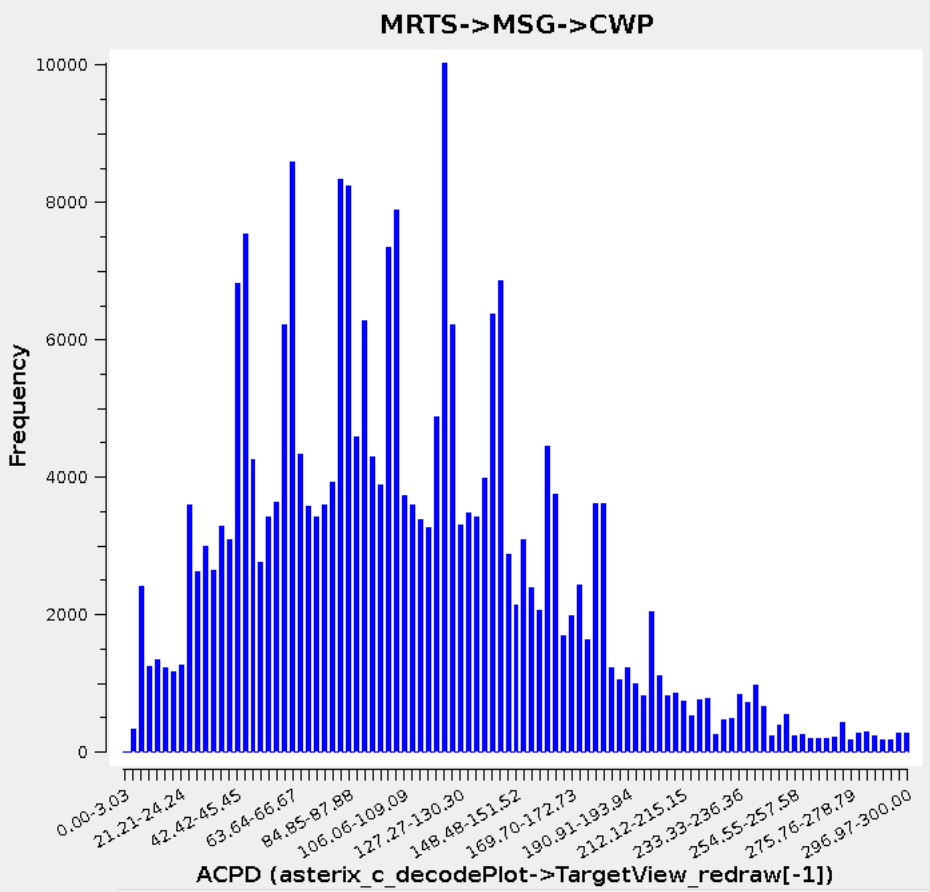
Image 9: The main transaction chain and the sub transaction chains

However, most of the computers used in ATC systems today are not real time systems therefore this requirement can not be an absolute hard limit. It has to be

statistically **significant**. The main aim of the following measurements is the inspection of this requirement in the different environments. It was found out during the tests that mainly 3 transaction chains are of greater interest. The main transaction chain (MRTS->MSG->CWP, marked blue in Image 9) lasts from the beginning of *MRTS::decodePlot* until the end of *CWP::TargetView_redraw*. The first child transaction chain (MRTS->MSG, marked red in Image 9) lasts from beginning of *MRTS::decodePlot* until the end of *MSG::TrackStore_updateTrack* including the communication between MRTS and MSG. The second child transaction chain (MRTS->MSG, marked green in Image 9) lasts from the beginning of *MSG::TrackStore_updateTrack* until the end of *CWP::TargetView_redraw*, including the communication between MSG and CWP.

8. Distributed test bed, live scenario

In the distributed environment the live scenario was further investigated. The heavy load scenario was not taken into account in this environment since the

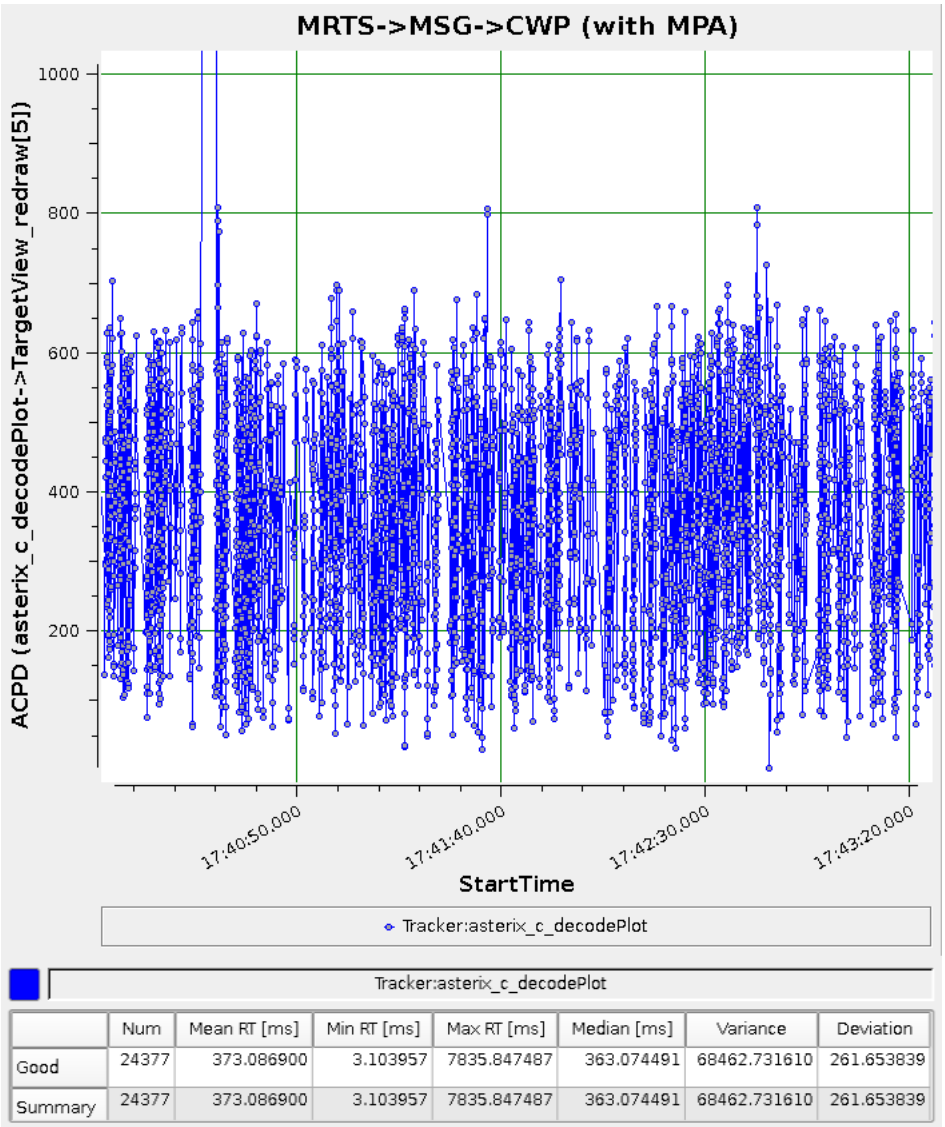


	Num	Mean RT [ms]	Min RT [ms]	Max RT [ms]	Median [ms]	Variance	Deviation
Good	277331	137.038157	2.658045	7835.847487	105.347920	52772.503121	229.722666
Summary	277331	137.038157	2.658045	7835.847487	105.347920	52772.503121	229.722666

Image 10: The main transaction chain with live data in the distributed test bed

amount of data would have been too great. 1 MRTS, 1 MSG and 110 CWPs were used for this test since 10 CWPs were shut down due to maintenance reasons. Image 10 shows the histogram and statistical parameters of durations for the whole processing chain (MRTS->MSG->CWP) of the live scenario in this environment, produced with the new frequency histogram visualisation and distribution parameter calculator of the tool. The analysis of the measurement data in this test bed shows the following results:

- Almost all processing chains fit into the “one second” performance requirement (mean duration 137.038 milliseconds with a standard deviation of ~229 milliseconds). To analyse the maximum duration values shown in Image 10 (Maximum ~7835 milliseconds) the command line tools were used to extract all durations and processed with statistical scripts. The result is that less than 0,3% of all chain durations are above one second. Moreover only 712 (~0,26% of the measurement population) single chain durations were above the 3-Sigma (792 ms) interval (3 times of the standard deviation around the median value). This fits exactly in the 3-sigma probability (99,73%) of normal distributed measurements and is therefore **statistically significant**. Last but not least the reason for high chain durations of a few measurements should be further investigated to get a better understanding of what is going on even in rare situations (such as CPU bottlenecks, waits in queues or network transmitting collisions for example).
- 110 of 120 CWPs were used so the amount of main transaction chains were multiplied by 110, leading to over 277000 measured chains.
- A minimum value of 2.7 milliseconds can be explained by NTP synchronization variations, i.e. the receiving computer has a time delay that is almost equivalent to the measured duration.



	Num	Mean RT [ms]	Min RT [ms]	Max RT [ms]	Median [ms]	Variance	Deviation
Good	24377	373.086900	3.103957	7835.847487	363.074491	68462.731610	261.653839
Summary	24377	373.086900	3.103957	7835.847487	363.074491	68462.731610	261.653839

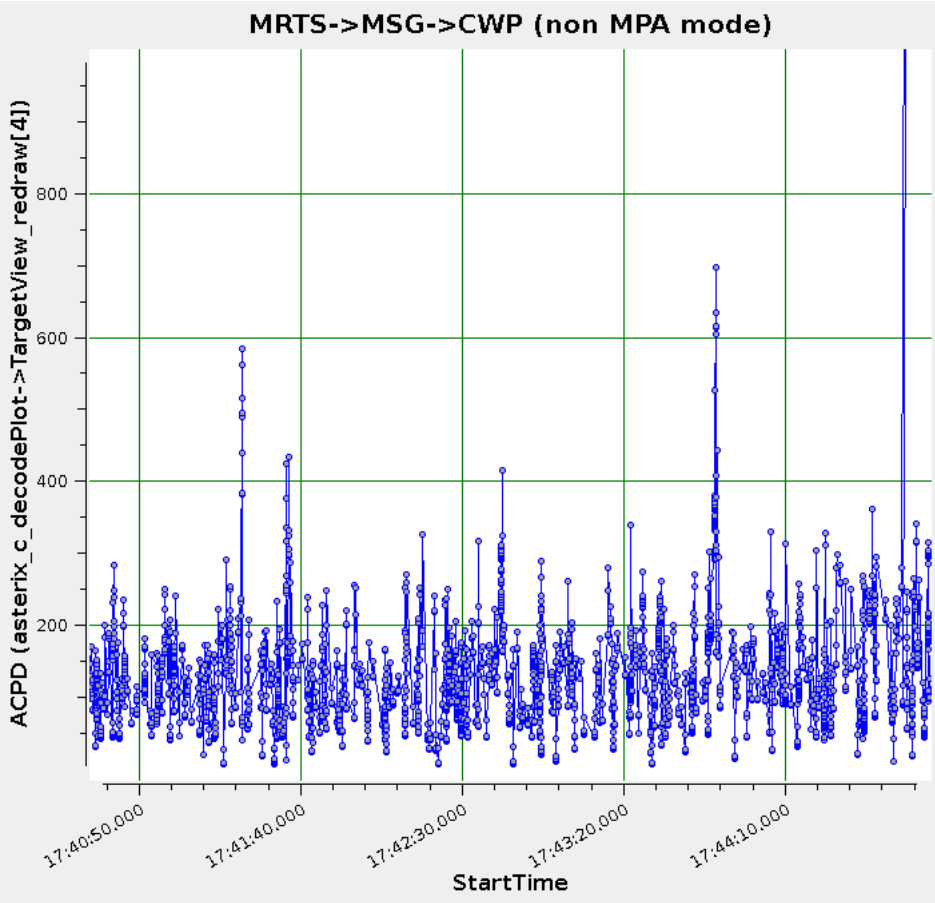
Image 11: The main transaction chain in MPA mode

The histogram shown in Image 10 was produced by investigating the ACPD chain duration between *MRTS::decodePlot* and *CWP::redraw* without taking the level of the *CWP::redraw* into account (i.e. the level was ANY).

It is also interesting to see the difference between the chain duration in normal and in MPA mode. To achieve this, two child levels had to be investigated for the statistics and curve computation, namely level 4 for normal mode and level 5 for MPA mode:

Level 4 (normal mode)	Level 5 (MPA mode)
MRTS::decodePlot MRTS::Correlation *MSG::UpdateTrack* *CWP::UpdateTrack* *CWP::redraw*	*MRTS::decodePlot* *MRTS::Correlation* *MSG::UpdateTrack* *CWP::UpdateTrack* *CWP::UpdateTrackMPA* *CWP::redraw*

The tool-generated scattergram in Image 11 shows the expected behaviour related to the chain duration, i.e. in MPA mode the mean duration with ~373 ms is much



	Num	Mean RT [ms]	Min RT [ms]	Max RT [ms]	Median [ms]	Variance	Deviation
Good	251205	114.389199	2.658045	7822.185730	101.440184	45672.194171	213.710538
Summary	251205	114.389199	2.658045	7822.185730	101.440184	45672.194171	213.710538

Image 12: The main transaction chain in normal mode

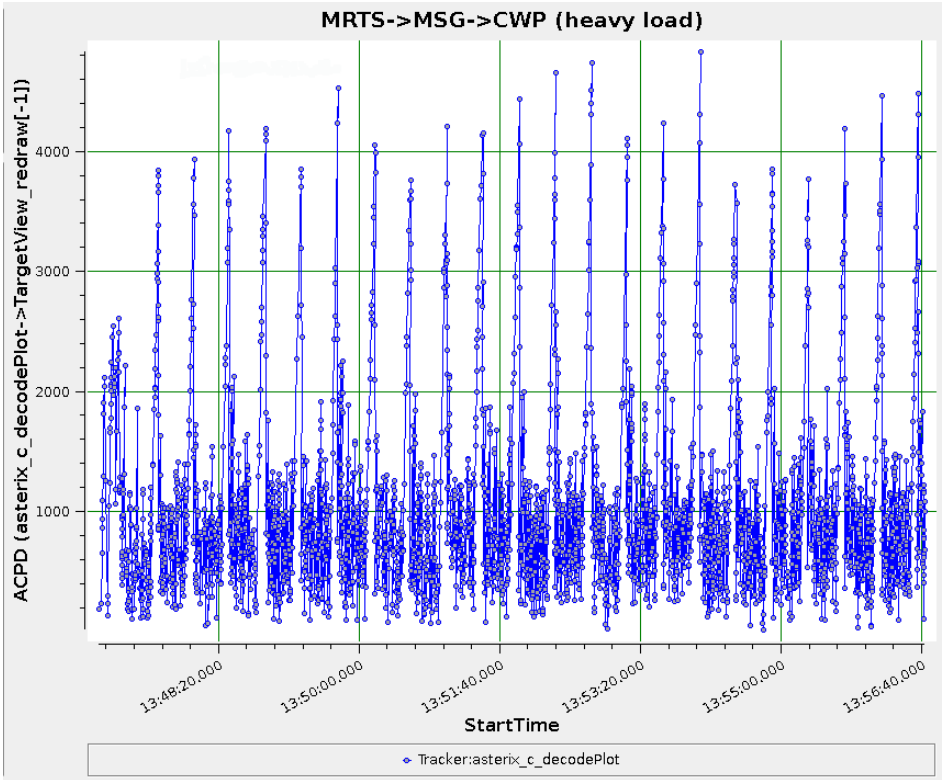
higher than the mean duration of the transaction chains in normal mode (~114 ms in Image 12). In this mode all received track updates are collected but not directly

processed to display (via *redraw* and *recalculate*). The tracks are drawn on the display timer driven every 500 milliseconds and not each time a track is updated.

This is the expected behaviour since the aim of the MPA mode is the optimization of CPU usage by drawing many targets with just one call instead of drawing each target individually. This aim can only be achieved by collecting targets for a certain amount of time thus increasing the duration of their travel time. The mean duration does not have a value of 500 milliseconds (the MPA mode forces a target draw at that rate) since targets are received in the whole 500 millisecond interval thus 500 milliseconds is the worse case for the waiting time in MPA mode for drawing a track update.

9. Standalone system, heavy load scenario (3000 tracks)

The next tests were made in the standalone environment but with the heavy load scenario with 3000 tracks. Image 13 shows the scattergram of the main transaction chain and its statistical values for that situation. The number of measured transactions is only 3126 because of the small scenario duration of 10 minutes. The mean duration is around ~1048 milliseconds and therefore almost 10 times higher than the mean duration in the live scenario. This high value can be explained by the fact that the computer where these values where measured is working with a very high CPU load, because MRTS, MSG and the CWP running on the same computer, thus consuming more CPU time. The CWP is trying to reduce CPU load by switching into MPA mode. However, even this mode is not successful enough to reduce the overall CPU consumption (due to the other processes on this computer) so the mean duration value is still much higher than the one measured in the live scenario above.



	Num	Mean RT [ms]	Min RT [ms]	Max RT [ms]	Median [ms]	Variance	Deviation
Good	3126	1048.425149	12.759335	4829.751732	845.781617	625835.171642	791.097448
Summary	3126	1048.425149	12.759335	4829.751732	845.781617	625835.171642	791.097448

Image 13: The main transaction chain and statistics with heavy load

Nevertheless the MPA mode is successful since the duration in non MPA mode increases rapidly when the CWP switches back to that mode. The CWP rechecks this decision every 25 seconds. Shortly after the mode switch the CWP again detects a higher CPU consumption so it switches back to the MPA mode for the next 25 seconds and so forth. Image 14 exemplarily shows for the second transaction chain (MSG->CWP) the transition between these modes for two adjacent peaks.

Summing up the findings of the evaluation of this scenario it can be said that

- The mean time needed to present a track update to a controller is above one second (1048.43 milliseconds) in this high load scenario.
- Maximum durations of more than 4 seconds occur caused by the CWP switching between standard mode and MPA mode. Usage of MPA mode shows the desired effects otherwise the CWP would not switch back to standard mode. Nevertheless the criteria to switch between these modes have to be checked.

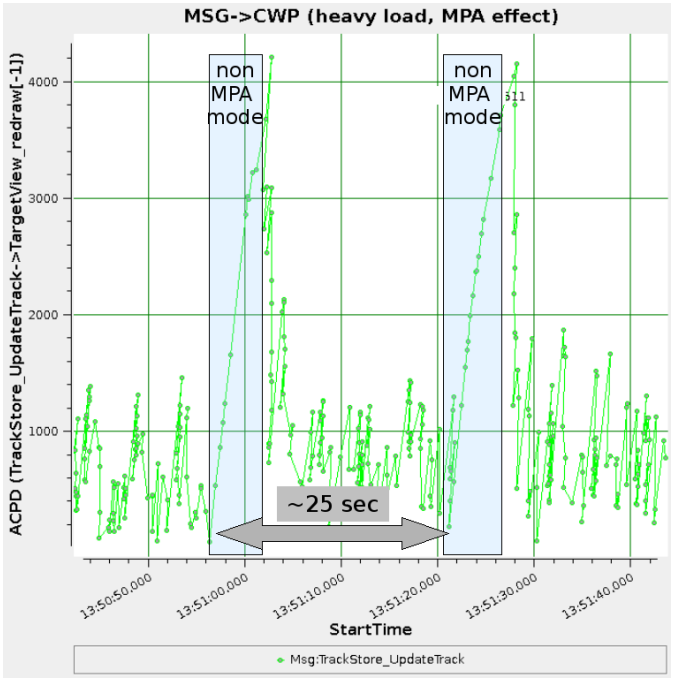


Image 14: The main transaction chain with heavy load

10.Summary and perspective

The ARM instrumentation of PHOENIX and resulting measurement data analysis provide a detailed insight view of the duration of the important processing steps of PHOENIX. Moreover, an overview could be generated of the whole processing chain starting from the point in time when a plot is received by the tracker until it is presented to the controller at the CWP. The main findings were:

- The mean duration needed to present a track update to a controller in an operational environment and under realistic conditions is with a statistically population of more than 99,7% (3-sigma interval) below one second.
- This is not true in a situation with a scenario of 3000 tracks running on one standalone computer. The mean value here is 1048 milliseconds. A wrong decision of the CWP to switch from MPA mode back to non MPA mode is responsible for high duration peaks up to 4 seconds thus also increasing the mean value. The aim here must be to improve the observed mode change behaviour of the CWP thanks to the observations made with ARM. This test can then be repeated with an optimized CWP.
- Other side effects were found which could not be presented in detail here. For example a mechanism that was introduced to minimize network load has the side effect that the mean overall duration is more than 200 times higher than the time the processes need for their internal computation. This effect is explained with process internal buffering of network packets before these are sent out on the network. It must be discussed how these buffering

algorithms in the different processes can be adapted to get a faster throughput.

A continued use of ARM in the software development process of PHOENIX is planned. A permanent performance monitoring of the system shall be introduced to control performance behaviour during the software development period with its various code modifications and its effects. Also semi-automatic ARM instrumentation would be helpful using a pre-compiler which can generate ARM code on the fly. This may be achieved by a modified MOC (meta object compiler) utility of the Qt® C++ framework using MyARM's QArm [Engels, K., Ruppert S. (2005)] framework.

In operational context a permanent performance management using ARM measurements can provide a “real-time” throughput information of single components and an end-to-end response time of the whole ATC system processing chain to the system management. At ANSP's the system management is in charge to control daily system opertaions and to provide first level support. This way the system management would have a chance to detect performance problems just when the system “starts to become slow”. Moreover, the information about the cause of a performance problem (e.g. a radar producing irregular data) would be very well known due to the measurement granularity.

11.Abbreviations and references

ANSP	Air navigation service provider	**MPA**	Multiplot Averaging
ARM	Application Response Measurement	**MRTS**	Multi-Radar Track Server
ASTERIX	All Purpose Structured EUROCONTROL Radar Information Exchange Format	**MSG**	Message Server
ATC	Air Traffic Control	**NTP**	Network time protocol
CWP	Controller Working Position	**RADNET**	Radar data network
DFS	Deutsche Flugsicherung	**WAN**	Wide Area Network
LAN	Local Area Network		

1) Euler, B.; Heidger, R. (2007); et al.: PHOENIX Systemhandbuch. DFS, Version 3.0, 20.6.2007
2) Heidger, R. (2006): A new generation RDP fallback system in the DFS. European Journal of Navigation, Sept. 2006.
3) The Open Group (2004a): Technical Standard: Application Response Measurement (ARM) Issue 4.0, V2 - C Binding, Berkshire, United Kingdom, ISBN: 1931624380
4) Ruppert S. , Engels, K. MyARM GbR website http://www.myarm.de
5) The Open Group (2007): Technical Standard: Application Response Measurement (ARM) Issue 4.1, V1 - C Binding, Berkshire, United Kingdom, ISBN: 1931624747
6) Engels, K., Ruppert S. (2005), QArm Tutorial, Version 1.0, MyARM GbR, Gelnhausen, December 2005