Integrated control system environment for high-throughput tomography

Igor Khokhriakov, Lars Lottermoser, Felix Beckmann

Igor Khokhriakov, Lars Lottermoser, Felix Beckmann, "Integrated control system environment for high-throughput tomography," Proc. SPIE 10391, Developments in X-Ray Tomography XI, 103911H (11 October 2017); doi: 10.1117/12.2275863

SPIE.

Event: SPIE Optical Engineering + Applications, 2017, San Diego, California, United States

Integrated control system environment for high-throughput tomography

Igor Khokhriakov*[a], Lars Lottermoser[a] and Felix Beckmann[a]
[a]Institute of Materials Research, Helmholtz-Zentrum Geesthacht, Max-Planck-Strasse 1, Geesthacht, Germany 21502

ABSTRACT

The extensive progress in hardware in recent years makes it now possible to develop nearly real time control system for tomography experiments. Such system can perform all the routines that are necessary for the experiment and provide real time feedback to the user. This feedback can be used for instant monitoring and/or for real time reconstruction.

The initial design and implementation of such system was presented in the SPIE publication in 2014 [1]. In this paper an update to the system is presented. The paper will cover the following 4 topics. The first topic simply gives an overview of the system. The second topic presents the way of how we integrate different software components to achieve simplicity and flexibility. As it is still in research and design phase we need a possibility to easily adjust the system to our needs introducing new components or removing old ones. The third topic presents a hardware driven tomography experiment design implemented at one of our beamlines. The basic idea is that a hardware signal is sent to the instrument hardware (camera, shutter etc). This signal is emitted by the controller of the sample axis which defines the moment when the system is ready to capture the next image i.e. next rotation angle. Finally, as our software is in a constant process of evaluation a continuous integration process was implemented to reduce the time cost of redeployment and configuration of new versions. Hardware triggering technique is used to achieve a fast and fully-automatized experiment.

Keywords: tomography, control system, data acquisition, continuous integration, hardware triggering

1. INTRODUCTION

In this paper we give an update of the development of our beamline's control system. In the article "Integrated control system environment for high-throughput tomography"[1] the system was described in detail so in this paper we mainly focus on the following topics: data flow management; continuous integration; hardware triggering and the collaborative effort of HZG with KIT on developing detector for the tomography experiments.

Data flow management appears to be very important topic as managing a big number of data sources is a quite challenging and error prone task if addressed directly in the control script. In our solution all the burden is removed from the beamline scientist and hidden into a dedicated component developed on top of Apache Camel. This component is responsible for building and maintaining data flow routes.

The next very important software related topic is *continuous integration* of all the components. Continuous integration is widely used technique [2, 3]. The main purpose of it is to automatize deployment of the software into the production environment. The point is that we have a dedicated meta project that assembles single package from all system's components and applies beamline specific configuration to the resulting package. The process is fully automatized so a new deployment can be done in a matter of minutes.

The next two topics covered in this paper - hardware triggering and the collaborative work - are more hardware related. *Hardware triggering* is an essential part of the setup to achieve truly fast performance experiments. In the corresponding section of this article a short overview of our setup is given. Basically, all the knowledge about current status of the experiment is possessed by the rotation stage so it can send hardware triggers to other hardware e.g. shutter. These triggers are transformed into software events consumed by the system. You will find a short overview of the in-house

Developments in X-Ray Tomography XI, edited by Bert Müller, Ge Wang, Proc. of SPIE Vol. 10391, 103911H · © 2017 SPIE · CCC code: 0277-786X/17/$18 · doi: 10.1117/12.2275863

developed detector at the end of the article. This electronic *detector* is designed specifically for the high-throughput tomography and can perform up to 5000 fps acquisitions.

2. INTEGRATED CONTROL SYSTEM ENVIRONMENT FOR HIGH-THROUGHPUT TOMOGRAPHY

In this section we will give a short overview of our in-house development of the integrated control system environment (X-Environment). This system was described in details in the article "Integrated control system environment for high-throughput tomography" [1] so, please, refer to that paper for more details. Here we will just describe the main components. High-level overview of X-Environment can be found in Figure 1 below.

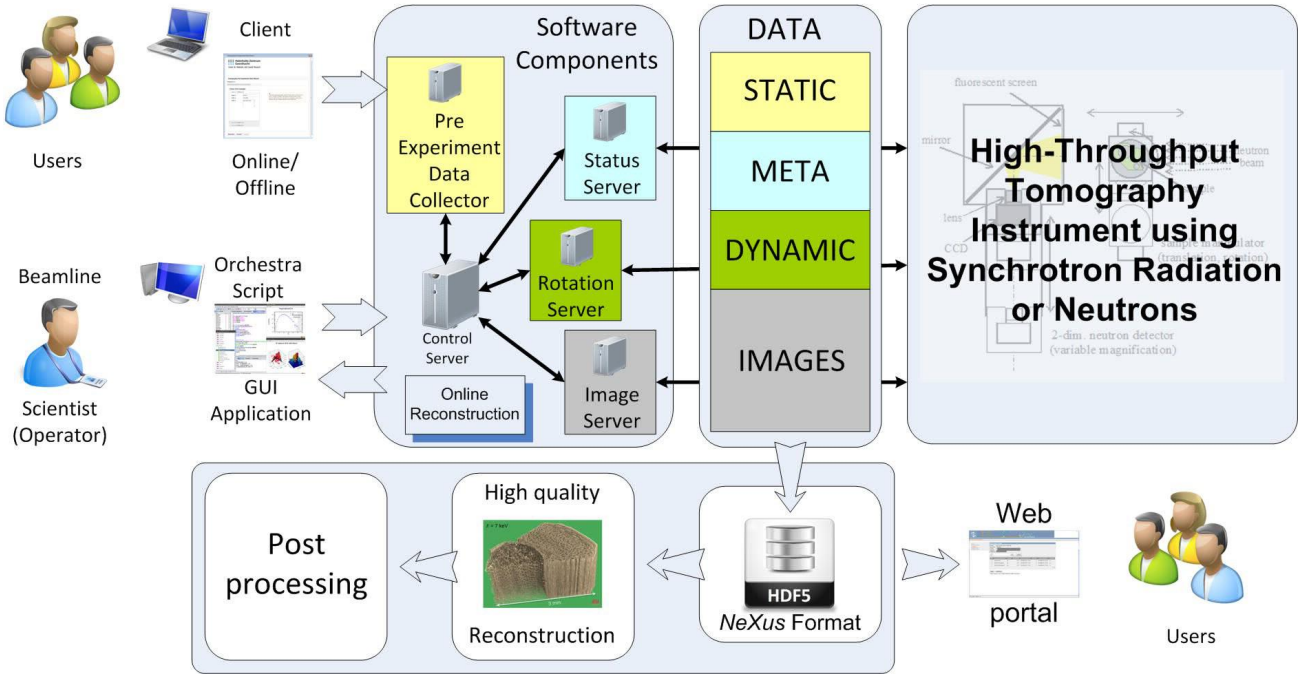

Figure 1. Integrated Control System Environment (X-Environment). This is in-house developed control system for our beamlines at PETRAIII. The system is based on TANGO meaning that all components are TANGO servers and talk via TANGO protocol to each other. The system also provides several user interfaces via web and IDL. The shown wood sample was one of the first measurements at P05 / PETRA III using part of the software infrastructure. Picture of the reconstructed tree structure is provided by Silke Lautner from Eberswalde University for Sustainable Development.

The components of X-Environment are:

- Status Server [4];
- DataFormat Server [5];
- PreExperiment Data Collector [6]
- IDL2TANGO bridge [7].

Status Server

This component is responsible for acquiring data from the upstream servers (for instance, instrument or storage ring hardware). The main features of this component are: aggregating data in a non-disturbing way i.e. it does not block the experiment if data is not available or can not be read from the upstream server; as yet another feature, it responses very fast to a main control process (the one which controls the experiment).

DataFormat Server

It is responsible for storing acquired data into a hdf5 file respecting NeXus format. NeXus structure can be defined using 3rd party tools (for instance [8]) or YAML [9].

As DataFormat Server may receive data from many sources we have defined a dedicated data flow management subsystem(see the next section).

PreExepriment Data Collector

This component provides intermediate storage of the data prior the experiment (during experiment all the data is written into hdf5 Nexus file via DataFormat Server). Users of our beamlines may use web interface provided by this component to specify their needs even before arriving to our facility. Beamline scientists may use this component for storing some typical configuration of the instrument.

Idl2Tango bridge

It is responsible for providing convenient TANGO abstraction in IDL programming language [10] for our beamline scientists. IDL is the main scripting language at our beamlines. Using this bridge beamline scientist can easily program logic like “wait until motor is not moving then do image grabing”.

3. DATA FLOW MANAGEMENT

Since the first introduction of the data flow management [1] we have adopted a new one. Comparison of these two designs conveys the advantages of the new set up:

Initial design

In X-Environment (see Figure 1) we have implemented a dedicated component (DataFormat Server) that is responsible for storing data in a hdf5 file respecting NeXus data format defined at DESY for synchrotron experiments [11]. Our initial work-flow is described in Figure 2 below. In our initial design beamline scientist controlled all the data flow to the DFS directly from an IDL script. This proved to be a quite unmanageable and error-prone solution. For instance, if NeXus structure had been changed beamline scientist had to go through all the scripts and adjust it accordingly.

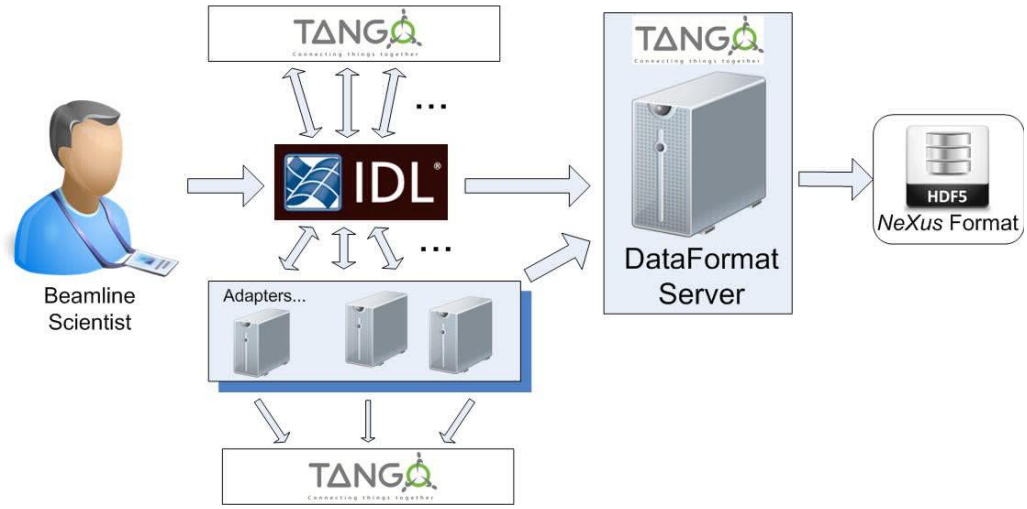

Figure 2. Data flow management, initial design: data is saved into hdf5 NeXus file via a dedicated component – DataFormat Server. Beamline scientist fully controls the process in an IDL script.

Revised data flow management

As pointed out in the previous section, the initial design of data flow infrastructure was suboptimal. So we came up with a new approach. The idea of this new approach is to implement data bus on top of Inter Process Communication (IPC) so that the beamline scientist has single entry point to the whole subsystem on a script level (Fig. 3).

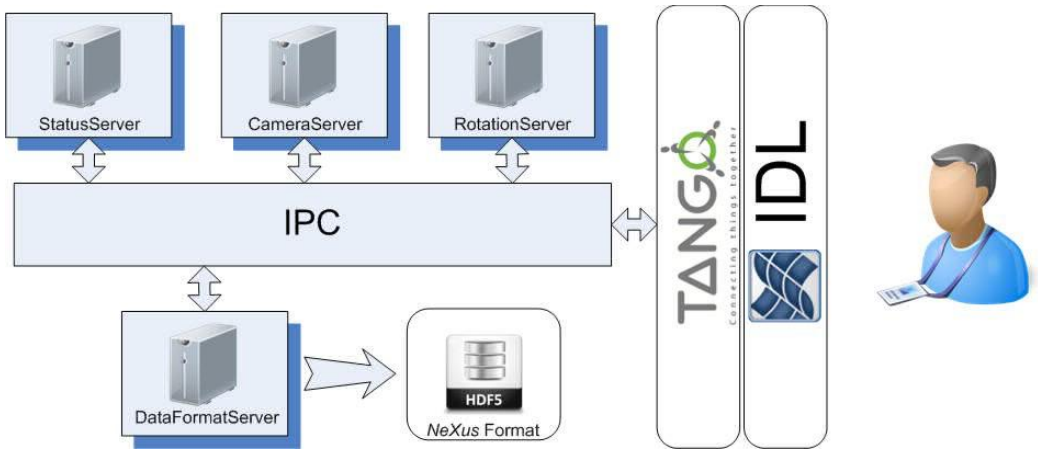

Figure 3. Data flow infrastructure using Apache Camel as IPC. IPC component provides a single entry point for the beamline scientist to data flow. All the data routes are hidden from the beamline scientist making the task of controlling data flow much easier.

To allow further customization and easier configuration of the data flow we have implemented a dedicated TANGO server that uses Apache Camel [12] for data routing [13]. All the endpoints in this case are defined in a simple xml configuration file shown below:

x-environment × | p05.properties × | p05.nxdl.xml × | p07.nxdl.xml × | routes.xml × | build.xml × | nxpath.mapping × | p07/.../routes.xml ×

```
<routes xmlns="http://camel.apache.org/schema/spring">
    <!-- here we define the bar route -->
    <route id="status_server_p07ct">
        <from uri="tango:p07/statusserver/p07ct?pipe=status_server_pipe&amp;poll=true"/>
        <to uri="tango:p07/dfs/0?pipe=pipe"/>
    </route>

    <route id="status_server_beamline">
        <from uri="tango:p07/statusserver/beamline?pipe=status_server_pipe&amp;poll=true"/>
        <to uri="tango:p07/dfs/0?pipe=pipe"/>
    </route>

    <route id="predator">
        <from uri="tango:p07/predator/0?pipe=pipe&amp;once=true"/>
        <to uri="tango:p07/dfs/0?pipe=pipe"/>
    </route>
</routes>
```

Figure 4. XML configuration of the Camel routes. Each route defines two endpoints. In this particular example data from two status servers and PreExperimentData collector is being transferred to DataFormat server.

This configuration file is hidden from beamline scientist and is defined once for every beamline (changes are possible afterwards though). With this setup beamline scientist controls data flow using very simple TANGO interface exported by TANGO Camel integration server. In fact there are only two commands “Start” and “Stop”. Even though it is still possible to send data directly to DataFormat Server from the script therefore providing extra flexibility for beamline scientist.

4. CONTINUOUS INTEGRATION

As our system dynamically evolves, the essential task is to setup a continuous integration pipeline. Specific feature of this pipeline must be the ability to easily change configuration of the target software package. All the components must be easily added or removed, for instance, when we switch from adapters to Apache Camel (see previous section). Apache Maven [14, 15] was introduced to achieve this goal. This is a dedicated meta project which, firstly, aggregates all the desired components into a single package and, secondly, applies specific configuration of a beamline to this package. This process is shown in Figure 5.

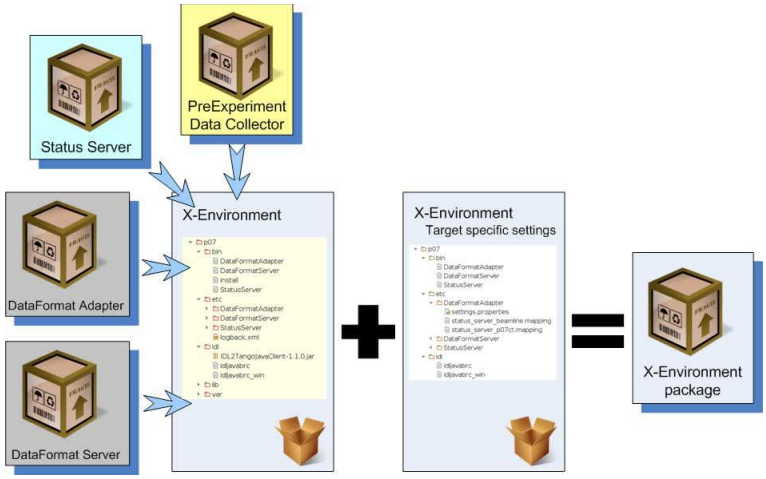

Figure 5. Single assembly of X-Environment. All the components are aggregated into a single package. Beamline specific configuration is applied to that package. The final result is ready for deployment.

Important note here is that every our component is a Java TANGO server, packed as a single executable jar file. Even components that require native code are implemented in Java using JNI [16]. In this case native libraries are also packed into the jar file. Therefore each component can be considered as a dependency of our meta project.

As this is a maven project we can simply indicate which components of which version we want to assemble in pom.xml file [15] as shown below:

```
m x-environment × | pom.xml × | effective-pom × | ezTangORB-1.1.14.pom ×
    <modelVersion>4.0.0</modelVersion>

    <groupId>hzg.wpn</groupId>
    <artifactId>x-environment</artifactId>
    <version>2.0</version>
    <packaging>jar</packaging>

    <properties>
        <!-- TODO move into a file -->
        <StatusServer.version>2.0.7</StatusServer.version>
        <!--<DataFormatAdapter.version>3.10</DataFormatAdapter.version>-->
        <CamelIntegration.version>0.3</CamelIntegration.version>
        <DataFormatServer.version>3.6</DataFormatServer.version>
        <PreExperimentDataCollector.version>3.0.5</PreExperimentDataCollector.version>
        <!-- other -->
        <xenv.root>${project.build.directory}${file.separator}${project.currentProfile}</xenv.root>
        <build.directory>${build.directory}</build.directory>
        <artifact.file.full_path>${build.directory}${file.separator}${project.currentProfile}.tar.gz</artifact.file.full_path>
        <artifact.file.name>${project.currentProfile}.tar.gz</artifact.file.name>
    </properties>
```

Figure 6. Maven's pom.xml fragment of the meta project. Under properties we define versions of the components that will be assembled into a single package.

Now as we have highly customizable but very well defined project, it is easy to automatize deployment procedure. The overall process is designed as follows: 1) all the components are deployed to maven repository; 2) CI server (TeamCity [17] in our case) fetches meta project from the repository, assembles it, executes integration tests and, finally, deploys assembled package to a production machine. This process is shown in Figure 7.

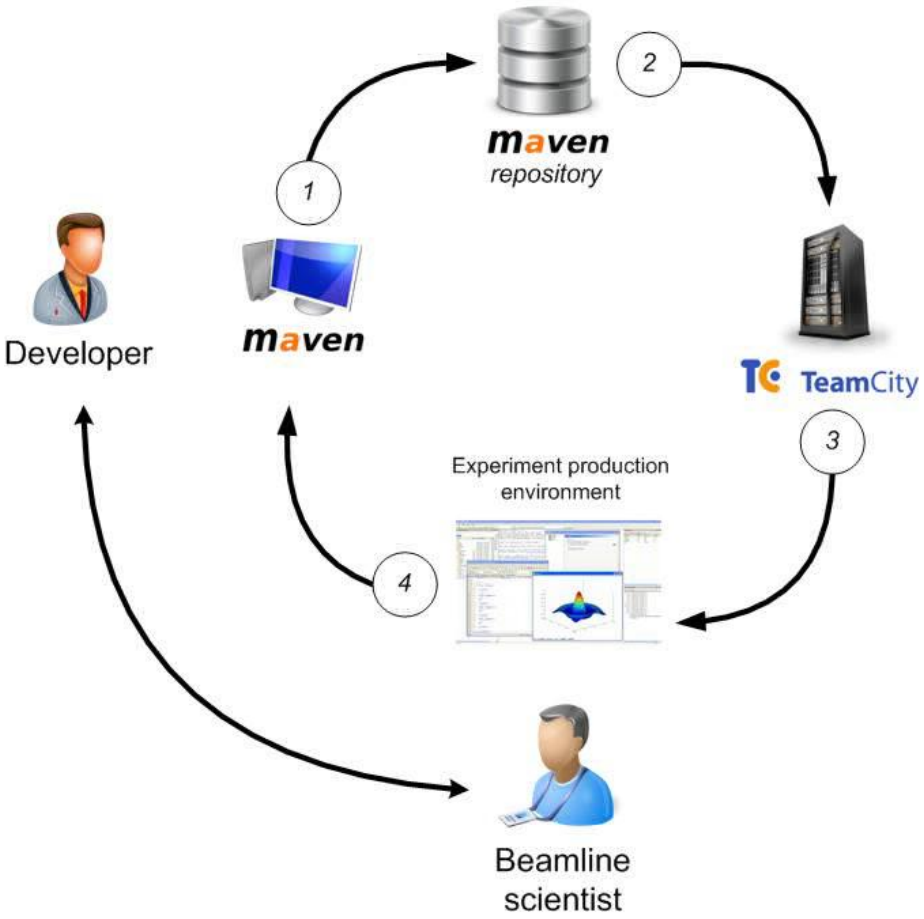

Figure 7. Software life-cycle in continuous integration process. First the developer releases software components to the maven repository (1). Continuous integration server fetches all the components via single meta project (2), assembles it and deploys to the production machine at our beamline (3). The beamline scientist provides feedback to the developer (4). The developer resolves the issues or implements a new component releasing it to the maven repository and cycle repeats itself.

If we need to change a version of component we can simply change corresponding property in pom.xml. To add a new component we simply add a new dependency to the project. In this way a new configuration can be deployed literally within one minute.

5. HARDWARE TRIGGERING

To achieve a fast fully-automatized experiment we are setting up a hardware triggering for our instrument. The idea behind this is straightforward: sample rotation hardware (Aerotech in our case) knows exactly when it is ready and it is the moment to capture the next image. So it sends a hardware trigger to an intermediate board (Zebra) which in its turn sends signals to the related hardware (camera, shutter, etc). Zebra module also sets a VME register which is polled by a specific software component. Once it catches a signal, this component sends an event to our DataAquisition sub system, hence, stores corresponding data snapshot to the end point (hdf5 NeXus file in our case). This process is shown in the diagram below.

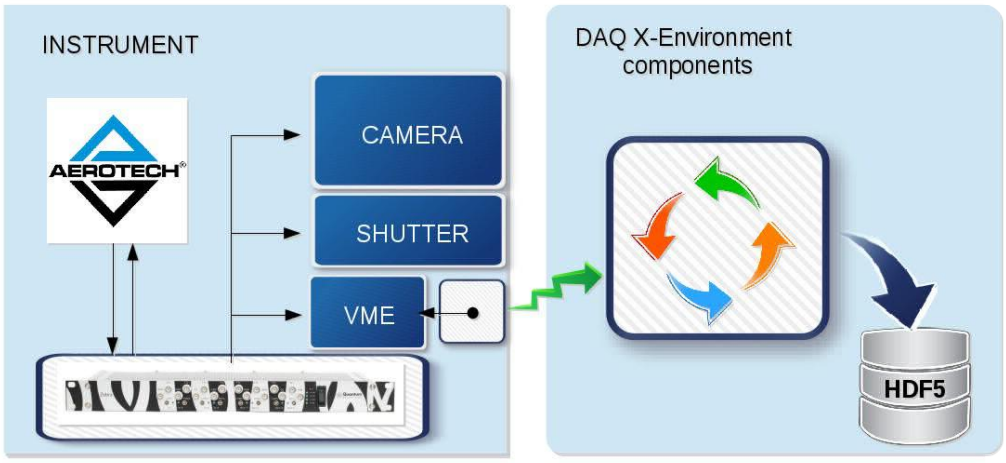

Figure 8. High level overview of the hardware triggering setup.

This setup not only speeds up the experiment but also greatly simplifies scripting as the only thing required from the beamline scientist is to specify a list of desired angles and pass it to the TANGO server that controls Aerotech. A dedicated process monitors the flow of the experiment and validates captured images.

As beamlines operating system for driving hardware at DESY is TANGO and for the storage ring is TINE [18] the most important feature of the setup is the ability to collect all required information from both TANGO and TINE layers in a non-disturbing way for the experiment. Furthermore, the acquired data accurately correlates to the timing provided by the rotation axis. This makes our setup being an unique integration solution between hardware and software components at our beamlines at DESY.

6. ULTRA FAST CAMERA

A collaborative effort with KIT on developing a detector for high-throughput tomography started in 2012. Today a working prototype is installed and used for the experiments at P05 and P07 (PETRAIII beamlines operated by HZG). Hardware and low-level software were designed and implemented at KIT [19]. Housing and cooling systems (Fig. 9) are designed and implemented at HZG.

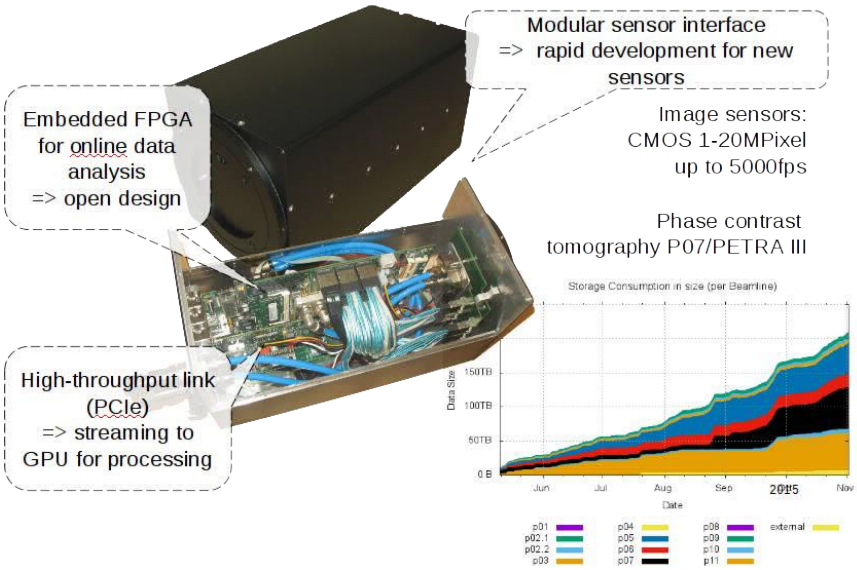

Figure 9. UCA camera – an in-house developed camera for high-throughput tomography.

This camera has flashable FPGA chip, so it can be easily reprogrammed to meet specific needs of the user. Modular sensor interface allows to switch sensor without need for replacing the whole camera. Another feature is a high data rate link which can be connected directly to GPU for performing initial analysis of the captured images.

Further in-deep overview of the camera performance in Phase-contrast tomography can be found in [20].

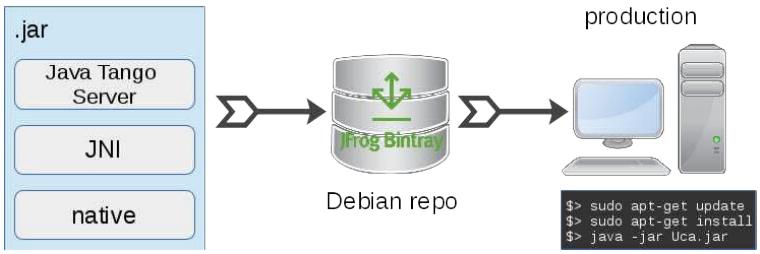

Figure 10. Uca camera Java TANGO server distribution. Camera server is packed into a single executable jar file with all native libraries embedded. This jar file is then packed into a deb file and deployed to our debian repository. The beamline scientist can get the package from it using standard debian tools.

At HZG we have implemented a high level abstraction for the camera – Java TANGO server [21, 22] – for easy integration with other our software. This server is being packed into a single executable jar file with all native libraries and therefore can be easily distributed to the production machine. In our case this jar is wrapped into a debian package, as we are using debian on our production machines, deployed to our bintray debian repo from where it can be easily fetched using standard debian tools. This process is shown in Figure 10.

7. CONCLUSIONS

A status update to our in-house developed beamline's control system environment is given in this paper. Main topics are: software progress in data flow management and continuous integration; hardware related configuration for achieving high-throughput tomography experiments.

X-Environment has reached its mature state and proofed to be a working solution at our beamlines P05, P07 at PERTA III in DESY, Hamburg, Germany. We are constantly exploring other work-flow solutions like Sardana or UFO project [23,24]. Due to the specific mixture of TINE and TANGO controls systems in DESY, neither of the frameworks provide the out-of-the box setup. Also we have our own requirements for acquired data and these requirements are not covered

by NeXus specification for tomography experiments. Efforts to adopt a new system based on an 3rd party solution would take too much time and men power.

Due to the fact that X-Environment is now used at our beamlines for the past few years, this paper covers our efforts on setting up continuous integration and increasing overall throughput by introducing hardware triggering and optimizing our in-house developed camera.

Some of the results achieved at our beamlines using X-Environment can be found, for instance, in this proceedings [25, 26]

Still there is some room for improvements. For instance, a new component that will aggregate all the logs from other components will have to be developed. A high level feedback could be then provided to the users based on the analysis of the collected logs via a web interface.

ACKNOWLEDGEMENTS

The Uca camera was developed within the Helmholtz project Detector Technology and Systems Platform (DTS) < http://www.helmholtz-detectors.de>. The concept of the software pipeline was a contribution to the ESS Design Update Phase.

A special thanks to the KIT employees involved in the camera's development, namely Michele Caselle, Matthias Vogelgesang, Suren Chilingaryan and Andreas Kopmann. They proofed to be perfect collaborators. They always provided (and still provide) immediate help in case of any problems with the camera's hardware and software libraries.

Authors would also like to express their gratitude to Silke Lautner from Eberswalde University for Sustainable Development for helping with the preparation of this paper and providing a picture with a reconstructed tree structure which is used in Fig. 1.

REFERENCES

[1] Igor Khokhriakov et al, “Integrated control system environment for high-throughput tomography”, Proc. SPIE 9212, 921217-1 (2014)
[2] Martin Fowler, "Continuous Integration", ThoughtWorks, <https://martinfowler.com/articles/continuousIntegration.html>, (2006)
[3] Continuous Integration, <https://en.wikipedia.org/wiki/Continuous_integration>
[4] X-Environment: Status Server, <https://github.com/xenvhzg/status-server>
[5] X-Environment: DataFormat Server, <https://bitbucket.org/hzgwpn/jdataformatserver>
[6] X-Environment: PreExperiment Data Collector, <https://bitbucket.org/hzgwpn/preexperimentdatacollector>
[7] X-Environment: IDL2Tango Java Bridge, <https://github.com/xenvhzg/idl2tango>
[8] Jan Kotanski, nxsconfigtool, <https://nexdatas.github.io/configtool/>
[9] Yet Another Markup Language, YAML, <https://en.wikipedia.org/wiki/YAML>
[10] IDL programming language, <https://en.wikipedia.org/wiki/IDL_(programming_language)>
[11] R. Gehrke, “Status Report on Workpackage 1 (Data Management)”,<https://indico.desy.de/getFile.py/access?contribId=1&resId=0&materialId=slides&confId=7333>, HDRI/PanData Workshop, DESY, (2013)
[12] Apache Camel, <http://camel.apache.org>
[13] X-Environment: Camel Integration, <https://bitbucket.org/hzgwpn/camelintegration>
[14] Apache Maven, <http://maven.apache.org/>
[15] X-Environment, <https://bitbucket.org/hzgwpn/x-environment>
[16] Java Native Interface, <https://en.wikipedia.org/wiki/Java_Native_Interface>
[17] TeamCity, <https://en.wikipedia.org/wiki/TeamCity>
[18] TINE, <http://adweb.desy.de/mcs/tine>
[19] Uros Stevanovic et al, “Ultrafast Streaming Camera Platform for Scientific Applications”, IEEE Trans. Nucl. Sci., vol. 60, no. 5, 3669 – 3677 (2013)

[20] Alexander Hipp et al, “High-resolution grating interferometer for phase-contrast imaging at PETRA III”, Proc. SPIE 10391, 10391-08 (2017)
[21] X-Environment: Java UCA camera server, <https://bitbucket.org/hzgwpn/uca>
[22] X-Environment: libUCA jni wrapper, <https://bitbucket.org/hzgwpn/libuca-jni>
[23] Z. Reszela et al, “Sardana - Scientific SCADA Suite”, Proc. ICALEPCS2017, FRBPL08 (2017)
[24] Vogelgesang Matthias et al, “UFO: A scalable GPU-based image processing framework for on-line monitoring”, Proc. of The 14th IEEE Conference on High Performance Computing and Communication & The 9th IEEE International Conference on Embedded Software and Systems , IEEE Computer Society, vol. 6, 824-829 (2012)
[25] Julian Moosmann et al, “Biodegradable magnesium-based implants in bone studied by synchrotron radiation microtomography”, Proc. SPIE 10391-0O (2017)
[26] Silke Lautner et al, “Using SRμCT to define water transport capacity in Picea abies”, Proc. SPIE 10391, 10391-18 (2017)