17th Int. Conf. on Acc. and Large Exp. Physics Control Systems   ICALEPCS2019, New York, NY, USA   JACoW Publishing
ISBN: 978-3-95450-209-7   ISSN: 2226-0358   doi:10.18429/JACoW-ICALEPCS2019-WEPHA060

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2019). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

FUTURE ACQUISITION ARCHITECTURE INVESTIGATIONS AT DIAMOND

K. Ralphs, J. Handford
Diamond Light Source, Didcot, UK

Abstract

At Diamond we are reviewing the current stack of in-house Software Applications that are used to control our beamline experiments and analyse the data produced by them. We intend to use this process of analysis and investigation to formulate proposals for a revised architecture to address the issues with the existing architecture, making use of the opportunities presented by modern technologies and methods, where appropriate. In doing so we hope to design a more flexible and maintainable system which addresses technical debt and functional limitations that have built up over the lifetime of our current software. This will allow us to go on to implement a powerful acquisition and analysis system to be used with the new facilities of Diamond II [1].

THE EXISTING DIAMOND SOFTWARE ENVIRONMENT

Diamond currently has a well-established stack of applications developed over many years which provides users with Data Acquisition and Analysis functionality from the Controls Hardware interface right up to the live and offline post processing and visualisation of experimental data. For organisational and technological reasons, the Controls, Acquisition and Analysis layers tend to have been developed by different teams, in some cases leading to hard boundaries of technological and operational knowledge. This often results in the need to find the expert from another team to progress the diagnosis of problems or obtain the knowledge of how to access useful information from a different layer of the stack.

Also, primarily in the Acquisition layer, the software has grown up organically over a period of 15 years leading to bad structure, un-needed redundancy, dead code and other forms of technical debt making it brittle, difficult to maintain and hard to develop. With the advent of the Diamond II redevelopment, we want to take a step back to analyse the software we have at the moment with a view to designing a new consistent platform architecture to address these and other problems and to take advantage of current industry best practises and technologies. In doing this the intention is to repackage the existing proven functionality in a more flexible structure behind a common platform API whilst revising some existing implementations and adding new capabilities and features along the way. This should allow us to migrate to a stable consistent platform that is easier to maintain and support and offers more flexibility to cross the old boundaries to get to the information required by the user.

CONCEPTUAL FUTURE ARCHITECTURE

This initiative is at a very early stage, though we do have a high level conceptual design on which we are focusing our technology examinations (see Fig 1).

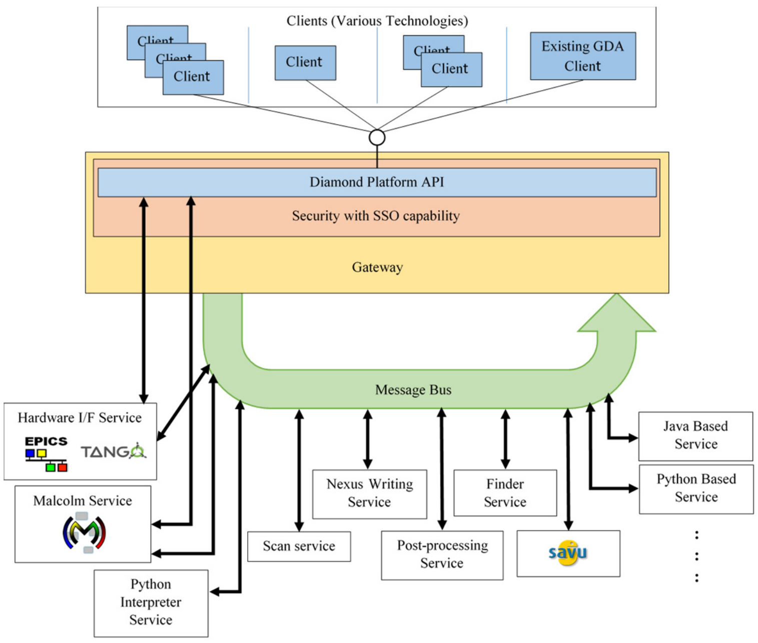

Figure 1: High Level Design.

Common Discoverable Reactive Platform API

The figure shows the adoption of a common 'Diamond Platform' API which would provide access to, and delivery of, all beamline control, data acquisition and data analysis functionality, from moving individual motors right up to reconstructing and segmenting tomography data for visualisation. Provision of an API like this will open up the possibility for new and custom clients to be developed for Diamond's experimental systems, so it is important that the choice of API technology is supported by many languages and environments. In addition, a common API will allow Diamond's software groups to address some of the challenges presented by the current architecture:

- **Effective Development Team Collaboration**. Using consistent technology fosters greater working between teams and simplifies maintenance and provisioning for IT support.
- **Secure Access**.
  The API will enable Diamond to put security at the centre of its software stack, allowing Single Sign-On (SSO) using the user's FedID and providing proper Authentication and Authorisation of requests.
- **Information Accessibility**.
  A common API covering all stack layers will facilitate reuse of elements and data from anywhere within the stack allowing higher level views to easily get to underlying information and structures.
- **Data Traffic Reduction**.
  Discoverability reduces data traffic whilst allowing clients to drill down interactively to gain more context for the headline data.
- **More Responsive User Experience**.
  Reactivity allows clients that need to, to operate interactively with live hardware, whilst also providing a means for feedback of progress of operations or availability of resources on a non-polled basis for any client(s) that can consume this.

Direct Communication for Time Critical Elements

At the Controls layer, it is important that no latency is introduced in the communication between the API and time critical elements like PandA boxes [2] and the Malcolm middleware layer [3]. At this level the API will be providing the logic design interface to these elements (as the current custom in-house developed API does) and so must be able to respond as quickly as possible to allow the current browser based Configuration Client to function properly. For this reason, it will be necessary for the Gateway to be able to route communications directly to the required consumer when appropriate.

Message Based Backbone with Services

Direct business layer access is not as strong a requirement for the Acquisition and Analysis layers; here requests can be serviced on a more command based timescale. The major issue to address in these layers is the high coupling (interdependency and intermingling) of not directly related areas of code; this is hindering efficient software development and compromising software maintainability. This is the classic problem of a monolithic application that has been developed over a long period of years by many people, where keeping the users' experiments running has often, necessarily, taken precedence over good software engineering practice. Despite this development constraint, we now have a very well tested application in Diamond's Generic Data Acquisition (GDA) software [4], [5] & [6]. GDA is well understood by users and beamline scientists and so it is important to maintain familiar proven functionality and continue to support well-established use cases. This means that we should be looking to preserve the algorithms that have been debugged and refined over the years, but present them in a more flexible and maintainable arrangement which strips out interdependency and encapsulates functionality in well-defined blocks that can be individually built, tested and deployed.

This is a familiar software architectural problem which is often addressed by adopting a message based micro-service architecture [7] to replace the old single application server approach. Microservices is a well-established pattern in the software industry these days and, though it has its own challenges (as all architectures do), there has been much development, debug and test effort invested in the approach by major organisations over the last 10 years resulting in a wealth of application and tooling to support such developments. In addition, adoption of this architectural pattern would open up possibilities unavailable in the current acquisition architecture, such as easy development of functionality in any language, service management to dynamically replace/supplement overloaded or failing functional elements, and application of containerized approaches for potentially streamlining deployment. For these reasons, this approach seems a plausible and attractive way of providing an infrastructure that would allow Diamond to gradually extract and repackage existing functionality from the current server, whilst maintaining its availability from the user perspective.

Testing the Plan

Having arrived at this design Diamond has now begun to examine the popular frameworks and available technologies to test if this candidate architecture can meet our expectations in terms of key requirements and our assumptions of the functionality they could deliver. We intend to use the outcome of these investigations to test whether such an approach is suitable for the many needs of Experimental Orchestration in the Synchrotron environment. If this is successful, it will then inform the decisions which lead to the choice of final technologies for the prototyping phase which would follow. As part of this analysis, we will necessarily need to evaluate candidate technologies and compare their pros and cons to see which,

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2019). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2019). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

if any deliver on enough of our key requirements and how viable and complex making up the deficit would be.

CANDIDATE TECHNOLOGIES

Because of the ubiquity of micro-service architectures these days, there are quite a number of potential frameworks and technologies in this field that offer good cross language support. In the area of the Platform API however, the choice is more challenging.

Platform API

The key requirements for an API technology to realise the Diamond Platform API are:

- **Adoption/Robustness**. It must be well tested and have been adopted at scale by major organisations.
- **Cross-Language Support**. It must be supported across at least Java and Python, but ideally have implementations in other languages too.
- **Discoverability**. It must support discoverability of the functionality it can provide. This both minimises the data traffic (since the API itself can be interrogated rather than having to ship lots of irrelevant content around) and facilitates higher level functionality finding out about the operations which serve it from lower level components.
- **Reactive**. It must support reactive methods, that is to say it must allow feedback to clients which subscribe to updates from server side functionality. This is a key requirement for the Controls interaction with Malcom and PandA, but it also facilitates extended functionality at the Acquisition and Analysis levels as it removes the need to poll for the status of operations such as completion of processing jobs and scan progress, further cutting down API traffic.
- **WebSocket Support**. It should if possible support communication over the WebSockets protocol as currently this is used in Malcolm and PandA interaction by the existing browser based Configuration Client.

Currently, the only technology that meets all these requirements is GraphQL [8] produced by Facebook. GraphQL was designed to be reactive and discoverable from the ground up as well as to address some issues of data transport volume with, for example, Representational State Transfer (REST). It has now been adopted by several large online organisations and has reference implementations in multiple languages plus dedicated IDE and Browser based tooling.

Gateway/Security Layer

Because of the prevalence of Web based interfaces and UIs, there are many frameworks in both Java and Python that provide easy realisation of gateway functionality incorporating SSO and OAuth2 [9] support. In the Java sphere the Spring Cloud Framework [10] is a strong contender providing high levels of standard functionality for very little developer effort with full SSO and authentication and authorisation support. In addition to this, support for Cloud based technologies such as load balancing and containerization, underpinned with a Message Based Micro-service architecture is also very easily achieved. This makes Spring Cloud a strong candidate given the rest of the planned design. Similar frameworks exist in python, but given that the vast majority of the user facing part of the API is likely to be developed by the Acquisition and Analysis teams, it is likely that a Java solution would be more suited.

Message Backbone

Here the choice is more open with many heavily tested cross language messaging frameworks in existence. Of these ActiveMQ [11] and Kafka [12] are strong contenders. Both of these can be configured as default options in Spring Cloud Micro-services installations and ActiveMQ is already in use in the Acquisition and Analysis teams providing an early service based implementation, currently limited in scope. Kafka offers other possibilities, being targeted at processing of data in transit and having the ability to replay message sequences in the event of system failures which could bring new capabilities to the Data Processing flows within the eventual system.

Other Supporting Technologies

Beyond these obvious contenders, other possibilities exist for the architectural layers identified and for other more low-level components which facilitate their use; for instance both Java and Python have well supported reactive programing library options which could be utilised to serve the needs of the API and will probably be used as appropriate depending on the most suitable language for a particular component.

CONCLUSION

The need to refresh the Diamond Data Acquisition Architecture is recognised as a key enabler to the smooth extension of data acquisition capabilities supporting Diamond II and in facilitating the adoption of GDA by other scientific institutions.

The roadmap for a Future Diamond Acquisition Architecture is at the conceptual stage, with a high-level outline approach having been proposed.

Currently the Software Groups at Diamond are assessing the technologies outlined in this paper together with other possible candidates and are using them to try to build demonstrator systems with mock services and clients to flush out all the issues and limitations of each across the various use cases. This will allow us to make a reasoned and validated set of decisions before committing major resource to the prototyping and development phases.

REFERENCES

[1] I. P. S. Martin and R. Bartolini, “Conceptual Design of an Accumulator Ring for the Diamond II Upgrade”, in *Proc. IPAC'18*, Vancouver, Canada, Apr.-May 2018, pp. 4046-4049. `doi:10.18429/JACoW-IPAC2018-THPMF008`

[2] S. Zhang *et al.*, “PandABox: A Multipurpose Platform for Multi-technique Scanning and Feedback Applications”, in *Proc. ICALEPCS'17*, Barcelona, Spain, Oct. 2017, pp. 143-150. `doi:10.18429/JACoW-ICALEPCS2017-TUAPL05`

[3] T. M. Cobb *et al.*, “Malcolm: A Middlelayer Framework for Generic Continuous Scanning”, in *Proc. ICALEPCS'17*, Barcelona, Spain, Oct. 2017, pp. 780-784. `doi:10.18429/JACoW-ICALEPCS2017-TUPHA159`

[4] Diamond Generic Data Acquisition (GDA) software, `http://www.opengda.org/`

[5] E. P. Gibbons, M. T. Heron, and N. P. Rees, “GDA and EPICS Working in Unison for Science Driven Data Acquisition and Control at Diamond Light Source”, in *Proc. ICALEPCS'11*, Grenoble, France, Oct. 2011, paper TUAAUST01, pp. 529-532.

[6] R. D. Walton *et al.*, “Mapping Developments at Diamond”, in *Proc. ICALEPCS'15*, Melbourne, Australia, Oct. 2015, pp. 1111-1114. `doi:10.18429/JACoW-ICALEPCS2015-THHB3O01`

[7] Martin Fowler and James Lewis, Microservices, a definition of this new architectural term, `https://martinfowler.com/articles/microservices.html`

[8] GraphQL open-*source* data query and manipulation language for APIs and runtime for fulfilling queries with existing data, `https://graphql.org/`

[9] *OAuth* open standard for access delegation, `https://oauth.net/2/`

[10] Spring Cloud framework for building robust cloud *applications*, `https://spring.io/projects/spring-cloud`

[11] ActiveMQ open source message broker, `https://`*`activemq`*`.apache.org/`

[12] Apache Kafka open-source stream-processing software platform, `https://kafka.apache.org/`

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2019). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.