Reference Architectures for Big Data Systems

JOHN KLEIN

MAY 22, 2017

Have you ever been developing or acquiring a system and said to yourself, *I can't be the first architect to design this type of system. How can I tap into the architecture knowledge that already exists in this domain?* If so, you might be looking for a *reference architecture*. A reference architecture describes a family of similar systems and standardizes nomenclature, defines key solution elements and relationships among them, collects relevant solution patterns, and provides a framework to classify and compare. This blog post, which is excerpted from the paper, *A Reference Architecture for Big Data Systems in the National Security Domain*, describes our work developing and applying a reference architecture for big data systems.

Last year, I worked with architects at the Data to Decisions Cooperative Research Centre to define a reference architecture for big data systems used in the national security domain. Acquirers, system builders, and other stakeholders of big data systems can use this reference architecture to

- **Define requirements**. The reference architecture identifies the architecturally significant requirements and discusses the architectural tradeoffs that are affected by particular types of requirements.
- **Develop and evaluate solutions**. The reference architecture identifies the modules that must be developed to realize a particular capability. Commercial off-the-shelf (COTS) or reusable technology can be mapped to particular modules in the reference architecture, providing a way to evaluate how the technology contributes to the solution and affects other solution elements.
- **Integrate systems together**. Existing systems can be mapped onto the reference architecture modules, allowing easier identification of overlaps and gaps and assessment of conflicting architecture decisions that might affect interoperation.

Scoping the Target Domain

We began by scoping the target domain. A reference architecture defines a family of related systems, and we know from our work in software product lines that scoping the target domain is a key to success. In particular, if your scope is too broad, the information in the reference architecture will be too general to be useful. If the scope is too narrow, however, the information will resemble the description of a single system and will not be easy for others to reuse. We scoped our reference architecture by defining a set of four use cases across a range of missions:

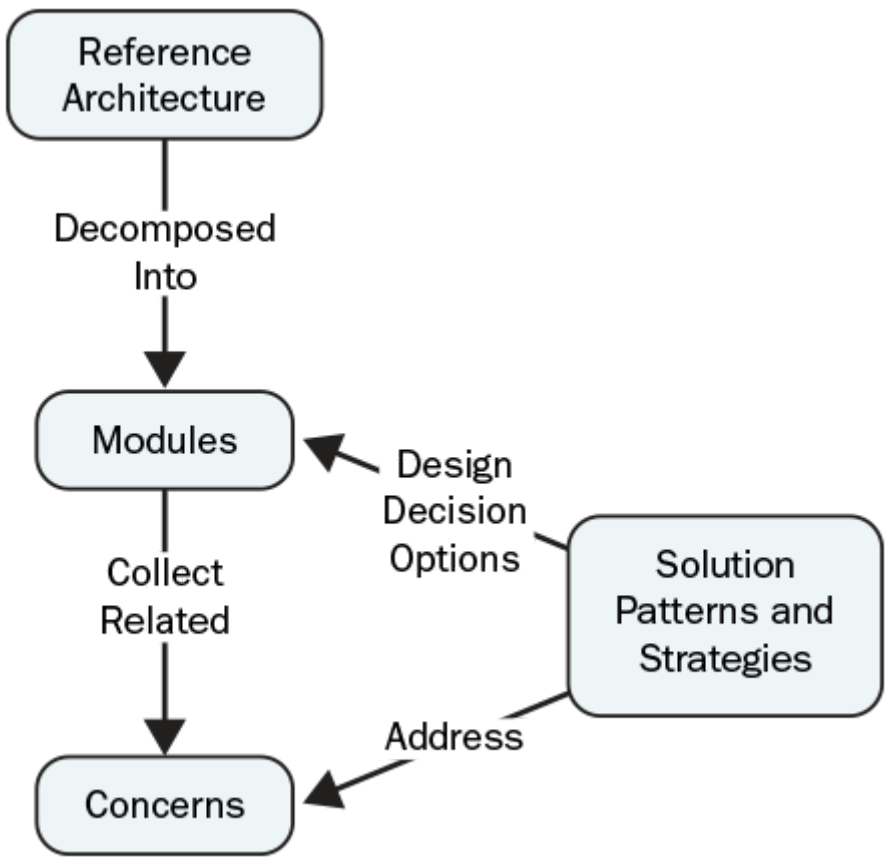
- strategic geospatial analysis and visualization

- full-motion video analysis
- open-source intelligence
- signals-intelligence analysis

From these use cases, we identified categories of requirements that were relevant to big data systems. These categories included data types (e.g., unstructured text, geospatial, and audio), data transformations (e.g., clustering, correlation), queries (e.g., graph traversal, geospatial), visualizations (e.g., image and overlay, network), and deployment topologies (e.g., sensor-local processing, private cloud, and mobile clients).

Our stakeholders had extensive experience developing and operating large-scale IT systems but needed help with the unique challenges arising from the volume, variety, and velocity of data in big data systems. We kept asking ourselves, *Is this type of requirement different in a big data system? If so, how is it different? What might a newcomer to the domain miss?* This analysis allowed us to reduce the background noise in the reference-architecture description, making the communication more effective.

We organized the reference architecture as a collection of *modules* that decompose the solution into elements that realize functions or capabilities and that relate to a cohesive set of *concerns*. Concerns are addressed by *solution patterns* (such as using the well-known pipes-and-filters pattern to process an unbounded data stream) or by *strategies* (which are design approaches that are less prescriptive than solution patterns, e.g., minimizing data transformations during the collection process). Together, modules and concerns define a solution-domain lexicon, and the discussion of each concern relates problem-space terminology (origin of the concern) to the solution terminology (patterns and strategies). Graphically, the model looks like this:



We found four types of concerns:

- **External requirements or constraints**. This concern captured external requirements or constraints on the system (e.g., type of workload), design decisions if we are considering an existing system (e.g., optimizations), or system quality attributes (e.g., latency and ease of programming). This type of concern has a significant impact on the design, analysis, or evaluation of a module.
- **Reuse and sharing modules**. This concern was related to reuse or ability to share modules. Examples included differences in execution triggers/rates (e.g., driven by input data streams, user requests, or repeated with a fixed period) and whether the functions or capabilities provided by a module were typically shared within or external to the big data system.
- **Stakeholder communities of interest and roles**. This concern aligned with stakeholder communities of interest or stakeholders' roles, such as processing

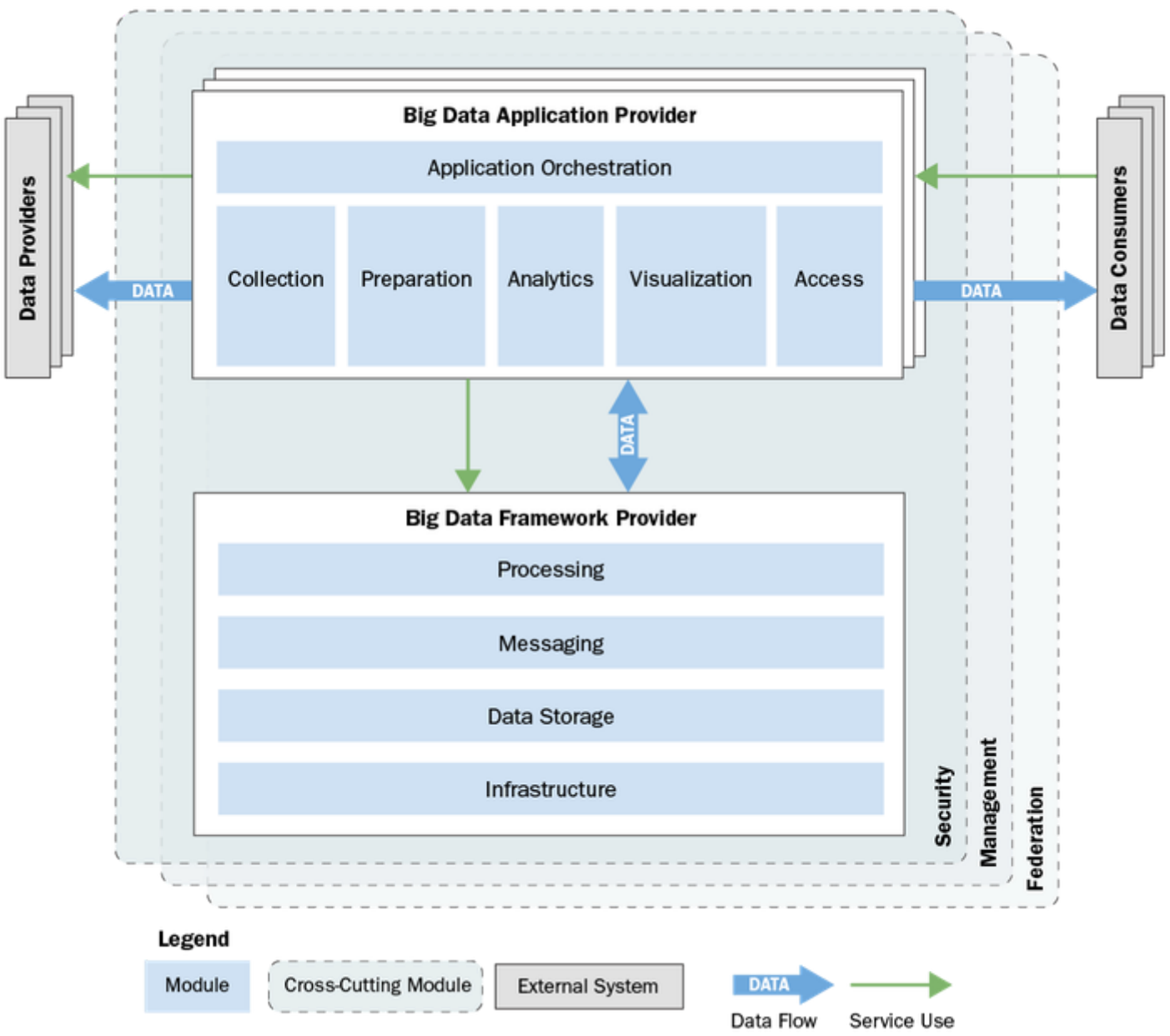
algorithms or system management, to explicitly include terms that would help stakeholders orient their perspectives on the reference architecture and identify the modules on which each stakeholder would need to focus.
- **Partitioning commercial and open-source big data packages and frameworks**. The last concern represented *de facto* partitioning of the commercial and open-source packages and frameworks that are used to realize big data solutions. Reflecting this partitioning in the module decomposition simplified the mapping between off-the-shelf technology and the reference architecture to help stakeholders position vendors and products within the reference architecture.

As noted above, we intended for this reference architecture to supplement other sources of general architecture knowledge. For example, while usability is obviously a concern in any human-computer interface, we did not specifically identify it as a concern in the reference architecture. In a big data system, however, providing an indication of data confidence (e.g., from a statistical estimate, provenance metadata, or heuristic) in the user interface affects usability, and we identified this as a concern for the Visualization module in the reference architecture.

Structuring the Reference Architecture

We used the four types of concerns described above to decompose a big data system into 13 modules grouped into three module categories:

- The **Big Data Application Provider** module category includes application-level business logic, data transformations and analysis, and functionality to be executed by the system.
- The **Big Data Framework Provider** includes the software middleware, storage, and computing platforms and networks used by the Big Data Application Provider. As shown in the figure below, the system may include multiple instances of the Big Data Application Provider, all sharing the same instance of the Big Data Framework Provider.
- The third module category is **Cross-Cutting Modules**. Each of the three Cross-Cutting Modules addresses a set of concerns that impact nearly every module in the other two categories. The complete module decomposition is shown below and described in detail in our paper.



In addition to the module decomposition, the reference architecture contained two supplemental sections to help our stakeholders apply the information. The first was a mapping that related COTS and open-source packages to the modules in the reference architecture. This simple tabular mapping allows a stakeholder to quickly understand how these technologies fit into the architecture--which solution capabilities each provides and how its use would affect the architecture of a system. A separate volume of the reference architecture maintains the mapping as it is the most dynamic and least normative prescriptive content..

We also returned to the use cases used to scope the reference architecture. For each use case, we showed how to use the reference architecture to design the architecture of a concrete system to realize the specified capabilities. In addition to providing a tutorial for our stakeholders, these examples served as an evaluation of the reference architecture contents and presentation.

If you are responsible for developing, integrating, or modernizing a number of systems that all deliver similar capabilities within a domain, creating a reference architecture can provide a framework for comparing, combining, and reusing solution elements.

Wrapping Up and Looking Ahead

This post (and our paper) describe a reference architecture for big data systems in the national security application domain, including the principles used to organize the architecture decomposition. This reference architecture serves as a knowledge capture and transfer mechanism, containing both domain knowledge (such as use cases) and solution knowledge (such as mapping to concrete technologies). We have also shown how the reference architecture can be used to define architectures for big data systems in our domain.

In the future, we would like to focus on the following areas of work:

- Using the module decomposition in the reference architecture to make decisions on where to standardize interfaces and implementations within a particular enterprise
- Creating new narrow and deep knowledge bases, similar to QuABaseBD or other modules within the reference architecture;
- Evaluating the utility of the reference architecture to define software product lines for sub-domains within the scope of the reference architecture

We welcome your feedback on this work in the comments section below.

Additional Resources

Read the paper on which this blog post was based, *A Reference Architecture for Big Data Systems in the National Security Domain*, which I co-authored with Ross Buglak, David Blockow, Troy Wuttke, and Brenton Cooper.

View my presentation *Runtime Assurance for Big Data Systems*.

Read the paper that I co-wrote with Ian Gorton *Distribution, Data, Deployment: Software Architecture Convergence in Big Data Systems*.

WRITTEN BY

John Klein

AUTHOR PAGE

MORE BY THE AUTHOR

Infrastructure as Code: Moving Beyond DevOps and Agile

JUNE 11, 2018 • BY JOHN KLEIN

Six Things You Need to Know About Data Governance

JUNE 19, 2017 • BY JOHN KLEIN

DIGITAL LIBRARY PUBLICATIONS ▸

SEND A MESSAGE TO JOHN KLEIN ▸

Three Roles and Three Failure Patterns of Software Architects

MARCH 21, 2016 • BY JOHN KLEIN

Big Data Technology Selection: A Case Study

FEBRUARY 8, 2016 • BY JOHN KLEIN

Model Driven Engineering: Automatic Code Generation and Beyond

MAY 11, 2015 • BY JOHN KLEIN

Get updates on our latest work.

Each week, our researchers write about the latest in software engineering, cybersecurity and artificial intelligence. Sign up to get the latest post sent to your inbox the day it's published.

Subscribe

Get our RSS feed

Carnegie Mellon University
Software Engineering Institute
4500 Fifth Avenue
Pittsburgh, PA 15213-2612
412-268-5800

2022 Carnegie Mellon University