The Art of Abstraction

HOW SEPARATING CODE, DATA, AND CONTEXT WILL MAKE YOUR BUSINESS BETTER

Silicon Valley Data Science
Continuum Analytics
Introduction

The traditional approach to supporting business with information technology has been via the enterprise application. Whether packaged systems for Customer Relationship Management (CRM), or custom-developed software to support specific lines of business, the application was the lens through which data was provisioned and architected.

Each application was written for a specific hardware platform and integrated specific off-the-shelf components, such as a relational database. This context for the data and the business logic was difficult to escape, as the need for performant systems drove architects to design and optimize for the specific features of those platforms. In today’s world, that tight coupling of architecture, data, and business logic hinders businesses, limiting their ability to combine data and analysis in new ways and for new business problems.

At this point the industry has practiced the separation of code from data for a while, but context remains sticky: one of the significant emerging challenges is the large number of different places that data comes from. It used to be that if you needed, for example, more data about your customers, there were two or three companies you could meet with. You’d sit down with them and explore your options, and then you’d receive a bunch of discs.

Now, data comes from APIs and many other channels, and is stored and processed all over the place, from FPGAs to CPUs to clusters of machines. There are many possible data structures, many possible sources, and many possible workflows. In an ecosystem like that, the code has to change every time the context changes—unless you can separate code from context with a layer of abstraction.
“If you construct an abstraction that pulls out code from its context, then you can write in a context-free way, and you can reuse your code more easily,” says Travis Oliphant, Co-Founder and CEO of Continuum Analytics. “Otherwise, you have to rewrite your algorithm just because you run it on a GPU instead of a CPU.”

Peter Wang, Co-Founder and President of Continuum, agrees that code reuse is a wonderful benefit of abstracting away the context. “But for me, that’s not as important as the idea that, if you have a foundation that insulates you from the details of how the computation is going to get done and how the data is stored—if you take those concerns off the table—then you’re left with a lot more freedom to talk about things that really matter: the insights I want to get from that data.”

The data you analyze, the code you use to analyze it, and the context in which that analysis is done all have to come together to achieve the insights that will grow your business. If you contemplate them separately, then you can develop them separately and create a set of building blocks that can be pulled apart and recombined in powerful ways.

**Schema-on-Read**

A canonical example of this strategy in action is the “schema-on-read” approach, which we’ll examine here as a detailed case in point. Silicon Valley Data Science recently implemented this technique for a healthcare company to isolate their patient management and processing systems from data variability from one insurer or provider to another, with great success.

**Schema-on-Write**

Thanks to the rigidity of historical database systems, the traditional approach to processing and integrating data has involved setting up a model for your data, or a schema, as the first step in designing your workflow. There are several good reasons for doing this. Schemas are useful for organizing database objects—and describing the relationships among them—and creating them early helps to maintain order and prevent interference in a relational database with many users, since there may be multiple schemas in one database. So traditionally, before any data even enters the system, the data structure has been set in place—this workflow is sometimes known as define, load, do. (Technically, it’s not even schema-on-write; it’s schema-before-write.)

Data from different sources goes through some ETL (Extract, Transform, Load) process to make it all uniform so that it fits the schema, enters the database, and is then used for whatever application was designed on top of that model. This is pretty efficient and works well up until the point where you want to run a second application off of the same data that requires the data in a slightly different schema. Then you’ve got to do another ETL process from one place into the next.

“If this schema was arrived at to support the first application (App1), the rest of them can suffer,” explains John Akred, CTO of Silicon Valley Data Science. “And so, what ultimately happens is that there’s inevitably some case in which App3 gets envisioned, and you look at App1 and App2 and say, ‘Oh man, you can’t get there from here. We just can’t build App3.’”
Then, rather than building a series of applications on top of custom-tailored schemas, you make the data dynamic on demand for various services. You could build a view and expose it as a service on Amazon, rather than building a direct, fragile link back to that database. You can also have multiple views or microversions of the same table. This allows you to define what you’re interested in on an application-by-application basis without dropping or losing data anywhere along the way. And you can still do transformations. There is architecture in between the database and the exposed view; if you wanted full name, you could pull first name and last name into the same view.

Road Test: Healthcare

In healthcare, one of the largest problems is that you have a thousand different players, each of whom describe the same problem slightly differently (as in the first name field example). With schema-on-write, non-conforming data can cause huge problems, the main one being that it may never get to the database in the first place.
Let’s say for the sake of argument that someone set the quality rules on the database to kick out any temperature value that’s outside the range of 90–102. As soon as someone begins reporting human temperature in Celsius instead of Fahrenheit, that would break in schema-on-write land, because you would have no place to store it if you haven’t defined it ahead of time. In schema-on-read land, however, we write the data no matter what. When you go to pick it up and take it to the application view, you may notice that it’s outside of your range, but you’ve stored it. So there’s much more resilience to provider differences and changes from your data sources.

Tools Built for Abstraction
In addition to techniques like schema-on-read, there are tools ready built with abstraction in mind. Continuum Analytics has been implementing this strategy of separating context from code and data in the software it creates: namely Blaze, Numba, and Wakari.

Blaze
Blaze (http://blaze.pydata.org/) provides a familiar interface to various computational resources. It integrates with existing SQL systems, NoSQL systems, the Hadoop infrastructure, and a number of ad hoc formats such as CSV files. It handles variable data formats with a uniform interface and creates a standardized set of expressions for connecting to existing tools to ensure interoperability. This means data scientists can use the same familiar and intuitive interface to analyze large and streaming datasets however and wherever they are stored.

“The goal was to build a system that could be equally agile over all of these pieces and to provide one uniform substrate to allow data analysts or data scientists to express their ideas,” says Wang.

Numba
While Blaze is about databases, Numba (http://continuum.io/developer-resources/numba) tries to do the same thing for hardware. Numba is a compiler designed to integrate with existing scientific Python software to convert a subset of pure Python code directly into optimized, low-level machine code without requiring the developer to think about the hardware platform where the code will be deployed. Numba allows data scientists to reap the clearer code benefits of a high-level language like Python, while also receiving the performance benefits of a compiled, low-level language. Numba can generate machine code for both CPUs and GPUs from the same high-level code. This provides a simple way for Python programmers to take advantage of new hardware resources without having to re-write their code.

Wakari
At the highest level of separating out context is a collaborative data analytics platform called Wakari (http://wakari.io). Wakari includes tools for exploring data, developing analytics scripts, collaborating with IPython notebooks, visualizing data, and sharing data analysis and findings.

The problem of sharing data analysis is not just figuring out a way to share an IPython notebook with someone; it also involves making sure that all the dependencies each notebook relies on, and all the data-files that notebook refers to, can also be migrated to any other machine. Wakari solves this problem by using the “package” and URI abstractions for the dependencies and the data respectively.
The Value of Abstraction

Schema-on-read is a fairly well understood design pattern. We may not go so far as to call it an industry best practice yet, but this process of abstracting away context does have distinct advantages.

Flexibility

“The traditional schema is purpose-built, and that makes it less flexible,” says Nanda Vijaydev, Principal Solutions Architect at Silicon Valley Data Science. If USPS wanted to add a new flat-rate package size, it would have to touch and modify about 20 different databases because the data modeling is so inflexible. There are lots of horror stories of situations where it would take six months to make a change like that and so companies just don’t ever do it.

For ecosystems where it can be highly beneficial to exchange a lot of data that is often in somewhat or slightly different formats important to the organization—such as in the healthcare example above—an approach that abstracts away the context can make integration at least tractable. And when you think of the number of integration scenarios the web provides, such as the Internet of Things and the fact that a massive number of home devices are going to be able to give some notion of status, or Amazon’s ability to integrate the inventory of several hundreds of thousands of suppliers, the flexibility that comes with abstraction isn’t just nice to have: it is the only way to make these systems possible.

Resiliency

“For me, the fundamental benefit is that it gives you resiliency that did not exist previously,” says Scott Kurth, VP of Advisory Services at Silicon Valley Data Science. “It’s resiliency in the sense that you can preserve information in case you need it, even if App1 doesn’t know anything about it.”

Simply writing all your data, rather than dropping it in the case that it fails to meet your pre-defined schema, allows you to achieve a resiliency in your data that was previously unapproachable. But it also makes possible powerful things like probabilistic integration. In metaphorical terms, this is essentially the idea that you separate the steps of hearing and listening. If you can write down what you heard so that later you can figure out what was said, that allows you to do be resilient.

To go back to the human temperature example: Say that you write down what you heard, which is one provider reporting human temperature in Fahrenheit and one reporting it in Celsius. Rather than depending on having knowledge of who reports in which scale, you can simply look and say, “If it’s lower than 70, it’s probably in Celsius.” It’s another part of our approach that adds up to extremely...
resilient but still flexible systems. Probabilistic integration allows you to work fast and smart, but in the event that you're wrong, you can go back and fine-tune your understanding of what was said because you've captured all the raw data.

**Lineage**

The ETL processes that happen in the traditional method of data integration often obscure the data source. You lose your connection to the input because you typically drop any staging tables once you've been successful in loading the data. Whether through an actual staging table or convoluted PL/SQL logic, it's very hard to get back to the source. But the way one often implements schema-on-read maintains that source and your ability to retrace your steps at a later date.

**What About Tradeoffs?**

This is all well and good, you may be thinking, but it can't all come for free.

Oliphant says that the trade-offs in implementing this type of abstraction do exist, but aren't relevant. “A good metaphor is the invention of compilers themselves, when we finally created functions and subroutines instead of having one big expression to run on the computer. You could ask that same question: Isn’t it going to be slower, because we have functions to call? Isn’t it going to be slower to write at a high level in C or Fortran because it won’t be assembly? And the answer is: technically yes, but the bottleneck problem is typically not the processing performance of the machine; it’s getting things written quickly.”

The same idea applies to the schema-on-read example. Such an approach does make your process more complex in getting from the database to an exposed view, because there may be 15 steps involved in transforming all the data. When we were building App1 with the old technique, it was a simple thing to do from our tidy database—if we presuppose that there will never be App2-AppN. If you know you’re going to have a lot of different uses for your data, and therefore you’re going to create a lot of individual pipelines, a schema-on-read approach may actually be more complex at first, but it will scale beautifully where the other approach does not scale well at all.

“Reducing complexity is never free,” says Kurth, “and so what we’re really doing is trading one type of complexity for another. We’ve added processing complexity in place of data complexity, but we’ve effectively traded unmanageable complexity for manageable complexity.”

**The Human Aspect**

If separating code from data from context is such an advantage, then why aren’t more companies doing it? As usual, the answer has a lot more to do with people than with technology.

Part of the problem is the “can’t see the forest for the trees” phenomenon at work. The client may be so close to the problem that they can’t step back and see that they have the same type of problem manifesting itself in fifty different places. They are paralyzed to recognize that the new data pipeline offers a way to condense the complexity into a small number of key places.
“The goal is to get you from an infinite customization problem to a single configuration problem,” says Kurth, “and that’s how you’re making complexity manageable.”

That can be very difficult to internalize. This kind of abstraction is so upside-down in the thought processes of people for whom traditional architecture is deeply ingrained that they struggle. The first step that they were taught from day one in the conventional way was to determine their schema, and then to map everything that they design in their database and everything that they design in the application to that schema. The rigidity of the approach makes them feel like they have more control. The schema-on-read approach feels very loosey-goosey to them.

But the flexible nature of these strategies is also the superpower of an abstract approach. It means that you can think in layers without having to do all the planning from day one.

“Separating code from context, separating code from data: it’s really about hiding one layer of detail and allowing people to talk about a different level of abstraction,” says Wang. “The creation of databases in general is the best example of this layered approach. Most people, when they write an SQL query against the database, they don’t know what kind of machine the database server is running on, they don’t know how many cores, they don’t know how many hard drives, they don’t know what optimization the IT or the operations people have done to it.”

Oliphant agrees. “Most people struggle because they have to worry about all the details. There isn’t a way to talk about them at a high enough level.”

Agility in Your Approach

Once you can talk about things at a high level and work with them at a high level, then you can iterate through various approaches quickly, Wang explains, allowing you to find the best approach to a given problem. “If the length of time it takes for you to get something done is so slow, then maybe the thing you finally produce in that development process is as fast as the machine knows how to execute, but you may still be doing the wrong thing.”

Agility is important, of course, not only at the development level, but also at the business level. “The time we find ourselves in right now is a really interesting moment where technology disruption and business disruption are both happening at the same time,” Wang continues. “The value for businesses is going to be in being very agile over that. The classic OODA (Observe, Orient, Decide, Act) loop, you can run that loop faster, and that’s what’s going to win.”

This is not always the case at all moments. There are times when the technology picture may be evolving, but
it's relatively stable. In those moments, it's a matter of fine-tuning what you've done to make it run faster than anyone else. But that approach matters most when you know generally what you're supposed to do and what direction you want to go in.

“But that's not where we are right now in this current moment of data science and business data and analytic disruption,” says Wang. “In the space we find ourselves in right now, where business value comes from turning data into insights, what's not seen are the business opportunities left on the table because you couldn't iterate fast enough.”

What about enterprise-scale businesses?
Both Silicon Valley Data Science and Continuum Analytics have seen a subtle shift beginning to happen with our enterprise-scale clients as individual lines of business and individual divisions recognize that they can take control of their own fate. More and more, these smaller organizational units are taking advantage of new technologies, rather than relying on the IT infrastructure to dictate what tools and techniques they have available to them.

Wang sees a business case to be made for allowing these kinds of minor rebellions. “If large corporations allow their individual lines of business to pursue that kind of data intelligence and market intelligence in their own way in a more independent fashion, then they don't have to go whole-hog, they don't have to convert everything. They can just try it. There's very little risk in doing that, and it's certainly better than doing nothing at all.”

This approach is what Edd Dumbill, VP of Strategy at Silicon Valley Data Science, has written about as The Experimental Enterprise—which he defines as “a company whose infrastructure is designed to make experimentation possible and efficient.” Part of that is an organizational mindset that values and facilitates investigation, and another part is a set of core technological capabilities such as open source software and cloud infrastructure.

According to Wang, the key to an experimental enterprise is keeping the technology stack lean and asking: What is the minimum amount of technology we need to deploy between the expert seeking the insights and where the data is? “By deploying more big iron enterprise software,” he says, “what you're really doing is you're putting more insulation layers between the data and the person who can have insights on data. And that completely displaces any possibility for agility.”

Conclusion
Ultimately, any tools or approaches that strive to separate code from data from context are meant to promote efficiency and to make impossible problems of volume, velocity, and variety possible to solve. The separation into layers of thought and building blocks of code provides a great amount of flexibility and agility that allow for rapid iteration and experimentation. By adding layers of abstraction, we can actually remove layers of technology—or at least shift them to a point in the process where they don't come between the human and the data. This allows businesses to adapt to changing market conditions, product needs, partner specifications, and vital decision-making scenarios.

“For all of our talk of data-centrism, the data is not the thing,” says Wang. “The thing is the human interaction with the data—the right humans interacting with the right data.”
About the Authors

Silicon Valley Data Science

Silicon Valley Data Science is a data science consulting company with a fast-growing team of experienced data scientists, engineers, architects, strategists, and designers. With experience in startups as well as enterprise-scale companies, we bring the best of both worlds to our clients: agility with an understanding of large-scale business.

Nanda Vijaydev
Principal Solutions Architect
With her strong data management and analytics expertise, Nanda brings a thorough understanding of Hadoop to information management and business intelligence solutions. Nanda uses her industry and technology experience to design practical and impactful big data solutions for clients.

Continuum Analytics

Continuum Analytics is focused on taking Python analytics, scientific computing, and data visualization to the next level. We have a simple purpose: better and faster answers, by keeping analysts and scientists in the driver’s seat. Whether computing on a enterprise data hub or a web cloud, the experience for the analyst should be equal.

Travis Oliphant
Cofounder & Chief Executive Officer
Since 1997, Travis has worked extensively with Python, most notably as the primary developer of the NumPy package, and as a founding contributor of the SciPy package. He is also the author of the definitive Guide to NumPy (2006). Travis engages customers, develops business strategy, guides the technical direction of the company, and actively contributes to software development.

Peter Wang
Cofounder & President
Peter is an active participant in the scientific Python and PyData communities. Author of the Bokeh web visualization library and the Chaco interactive visualization toolkit, he has extensive experience developing flexible, performant analysis applications and environments across multiple domains, including finance and high-frequency trading.

John Akred
Chief Technology Officer
With over 15 years in advanced analytical applications and architecture, John is dedicated to helping organizations become more data-driven. He combines deep expertise in analytics and data science with business acumen and dynamic engineering leadership.

Scott Kurth
Vice President, Advisory Services
Building on 20 years of experience making emerging technologies relevant to enterprises, Scott crafts vision and strategy for organizations. With a background in architecture and engineering, he combines deep technical knowledge with a broad perspective, to focus on business value.

Julie Steele
Director, Communications
Julie thinks in metaphors and finds beauty in the clear communication of ideas. She is particularly drawn to visual media as a way to understand and transmit information, and is co-author of Beautiful Visualization (O’Reilly 2010) and Designing Data Visualizations (O’Reilly 2012).