Identity by Any Other Name

THE COMPLEX CACOPHONY OF INTERTWINED SYSTEMS

PAT HELLAND, SALESFORCE.COM

*“What’s in a name?
That which
we call
a rose by
any other name
would smell
as sweet.”
—William
Shakespeare
[Romeo and Juliet]*

As distributed systems scale in size and heterogeneity, increasingly they are connected by identifiers. These may be called IDs, names, keys, numbers, URLs, file names, references, UPCs (Universal Product Codes), and many other terms. Frequently, these terms refer to immutable things. At other times, they refer to stuff that changes as time goes on. Identifiers are even used to represent the nature of the computation working across distrusting systems.

The fascinating thing about identifiers is that while they identify the same “thing” over time, that referenced thing may slide around in its meaning. Product descriptions, reviews, and inventory balance all change, while the product ID does not. Reservations, orders, and bookings all have identifiers that don’t change, while the stuff they identify may subtly change over time.

Identity and identifiers provide the immutable linkage. Both sides of this linkage may change, but they provide a semantic consistency needed by the business operation. No matter what you call it, identity is the glue that makes things stick and lubricates cooperative work.

This article is yet another thought experiment and rumination about the complex cacophony of intertwined systems.

THE NEED FOR IDENTITY

For a long time, we worked behind the façade of a single centralized database. If you wanted to talk to other computers, that was an “application problem” and not in the purview of the system. Data lived as values in cells in the relational database. Everything could be explained in simple abstractions, and life was good!

Then, we started splitting up centralized systems for scale and manageability. We also tried to get different systems that had been independently developed to work together. That created many challenges in understanding each other⁴ and ensuring predictable outcomes, especially for atomic transactions.

As time moved on, a number of usage patterns emerged that address the challenges of work across both homogeneous and heterogeneous boundaries. All of those patterns depend on connecting things with notions of identity. The identities involved frequently remain firm and intact over long periods of time.

Data on the outside vs. data on the inside

In 2005, I wrote a paper, “Data on the Outside versus Data on the Inside,”⁷ that explored what it means to have data not kept in the SQL database but rather kept in messages, files, documents, and other representations. It turns out that information not kept in databases emerges as immutable messages, files, values (à la key/values), or other

representations. These are typically semi-structured in their representations, but *they always have some form of identifier*.

Scale, long-running, and heterogeneous

Systems are knit together by identity, too. As homogeneous solutions are designed for scale, shards, replicas, and caches are all based on some form of identity. Solutions respond to stimuli over time, using one or more representations of identity to figure out what work to restart or continue. Connecting independently created systems with their own private and distrusting implementations always uses shared identities and identifiers that are the crux of their cooperation.

Searching and learning

Many other parts of the computing landscape depend on identities. Searching assigns document IDs and then organizes indices of search terms associated with them. Machine learning binds attributes with identities. In many cases, a set of attributes becomes interesting and is then assigned an identity. The system repeatedly works to associate even more attributes to them. It's when these attributes form patterns across the identities that the machine has learned something.

Identities: The new fulcrum

Computing patterns show our dependency on identities. We used to look only at relational databases but now we see pieces of computation and storage interconnected by identities. The data and computation connected by

identities can swirl and shift around.

The identifiers connecting these pieces remain immutable while the stuff they identify spins and dances and evolves. Similarly, whatever is using the identity may be simply a mirage while the identifier used remains solid.

What's in a name?

This article refers to *identities*. There are an astonishing number of synonyms for identity. All that really matters is that the identity is unique within the spatial and temporal bounds of its use. Name, key, pointer, file name, handle, check number, UPC, UUID (universally unique identifier), ASIN (Amazon Standard Identification Number), part number, model number, SKU (stock keeping unit), and more are unique either globally or within the scope of their use. It is the immutable nature of each identifier *within the scope of its use* that allows it to be the interstitial glue that holds computation together.

USING IDENTITY TO SCALE

Identity may be used to scale homogeneously and heterogeneously. This section examines a very complex example. Ecommerce not only uses shopping carts and scalable product catalogs, but it may also derive product descriptions by combining the best information from some of its many merchants and manufacturers. This information may be identified by merchant SKUs or manufacturer part number. In addition, inventory, pricing, and condition of offered goods all vary by merchant and are identified

in a nonstandard way. Many different connected and disconnected identities weave through the complex multi-company ecommerce.

Session-state and shopping carts

Each shopper gets their own shopping cart. This can be associated with an online account or with the web session. Shoppers don't get multiple shopping carts during a single web session. Furthermore, no one expects or wants the shopping cart to share state or consistent updates with other shopping carts.

The uniqueness of the shopping cart is provided by the shopping cart ID. There's some logic in the system to bind the session, either via user login or online session state, to a shopping cart ID. Based on that unique ID, the shopping cart contents are located.

The scalable key-value store

One common pattern in scalable solutions is the scalable key-value store. Take, for example, an ecommerce retail product catalog. The retailer has a whole bunch of products, each with a product identifier. The product description cache is sharded by the product ID. This supports scalable description data. Replicated shards support scalable read traffic. To add more product descriptions, add more shards. To support more read traffic, add more replicas of the shards. See the scalable catalog of product descriptions indexed by the product ID in figure 1. There's no requirement that the product catalog can update different products atomically. In fact, the product catalog cannot update all the cached entries

for a single product atomically!

Identifying cached jittery versions

Updates to product descriptions distribute new versions to replicas over time. Hence, reads are jittery, and later reads may show earlier values. Product ID is the immutable glue

FIGURE 1: A SCALABLE CATALOG OF PRODUCT DESCRIPTIONS

that makes this work. Even if the read of the cache returns an old cached value, it is associated with the desired product ID and meets the business needs. In product catalogs and for many other uses, old values are fine.

Matching and deriving descriptions

In most large ecommerce sites, product descriptions come from data submitted by manufacturers, merchants, and other sources. To correlate these, it is necessary to normalize inputs, match descriptions from different sources, and then combine them to get the best information available. Inputs arrive with identifiers such as model number, UPC, and SKU, defined by the third-party merchant selling through the large ecommerce site. *There's no single identity before matching.*

Normalizing cleans up the various inputs to try to have a consistent representation. If the color is Kelly green, forest green, olive, or chartreuse, should it be normalized to *green*? Normalization makes it easier to match various inputs to each other. It also loses some of the fidelity of the original input.

Matching attempts to find stuff that is the same. Is this product for sale from Merchant A the same as another product for sale from Merchant B? Each merchant has its own SKU as a personal unique identifier. How can they be correlated?

Slippery and sliding identifiers

Another challenge is that the merchants' SKUs are assigned and bound by the merchants. There's nothing to stop them from changing SKU 12345 from a pair of ruby

slippers to a can of chocolate sauce. *When your partner business uses identifiers in a non-immutable way, you need to be on your toes.* I've heard tales of small merchants with 40 bins of stuff in their basement. The contents of SKU #23 corresponds to whatever product is kept in bin #23 at the time.

UPCs: The same but... maybe different!

Consider large retailers that consolidate many sellers' goods through the large retailer's platform. It's helpful if the merchants have the UPCs in the description of their item[s]. UPCs make it much easier to match items from different merchants. Each of these 12-digit identifiers is for a *particular manufactured product*. The UPC works along with the EAN-13 (European Article Number 13) code, which is a bar code supporting scanners mostly for retail environments.

UPCs are *mostly* correct. Achieving consistency and equivalence of products with the same UPC is hard for both manufacturing and retail. Not everything has a UPC. Hand-crafted items, for example, may not have UPCs. For a number of years, shoes were notorious for not having UPCs.

Books: ISBNs, paperback, used, and digital

What about books? The ISBN (International Standard Book Number) is a 13-digit (formerly 10-digit) number that uniquely identifies a particular *version and format* of a book.

What about reviews? Most reviews are about the contents of the book, not the quality of the paperback's

binding. Don't you want to have shared reviews for the ebook, paperback, and hardback editions? Typically, this is handled with yet another unique identifier used to represent all the different versions and formats. Similarly, many times the same online products share reviews when the color and unique identifier differ.

A tangled web: products, SKUs, offers, inventory, and shippability

Online retail is an ocean of unique IDs, all weaving across different systems, concepts, and cooperating companies. Merchants will describe *their perspective* of goods for sale as their SKUs. Matching and correlating these goods into products from the perspective of the ecommerce site is a major endeavor in data science and machine learning. When done, the correlation is kept to facilitate working across the merchant and ecommerce site. Of course, the merchant is free to label a completely different product with the same SKU tomorrow; the ecommerce site must adapt.

The identifiers for products will reference the product catalog. The contents of the product catalog will evolve and be cached for efficient scalable reads. When accessing the cache, it may race with updates to the cache, and later reads may return earlier versions of the product description. It doesn't matter because either version is OK. The product catalog does not need transactional consistency.

Next, an offer to buy from a merchant is presented. Do you want a new or used product? What condition is it in, and what's the reputation of the seller? These offers are

FIGURE 2: **ECOMMERCE – A TANGLED WEB OF IDENTIFIERS**

correlated to the product, the shopping cart, the inventory for the specific offer, the price, the shipping commitment, and the details of how it will be shipped. Of course, this needs to be tied to the payment.

Each of these relationships across internal and external systems is knit together using various related identities. Figure 2 shows a very small subset of these interactions and how identifiers knit them together. Oh, yeah, the

ecommerce retailer hopes the merchant hasn't recycled the SKU when an order is placed. Attaching the product description to the SKU usually avoids confusion.

USING IDENTITY TO SEARCH

Let's consider web search as we've all seen it in Yahoo!, Google, and Bing. Not surprisingly, searches are accomplished by assigning unique IDs to each of the documents in the web.

Document IDs, URLs, and search terms

As these huge web crawlers traverse the URLs they find to locate documents, they remember the URL for each document. These URLs form unique IDs. It's common to bind the URL to another unique document ID that's shorter.

As the document is crawled, the word sequences are extracted for indexing. These word sequences (known as N-grams) correspond to the search terms entered into the web search application.

N-grams are sharded into a large number of partitions. As multiple search terms enter a search, the shards that may hold those terms are queried. This returns sets of document IDs from many shards. By comparing the results looking for document IDs in common across the search terms, a resulting collection of document IDs can be returned.

While this is vastly and grossly oversimplified, the main point is that search is all about identities.

Searching an object-relational app

Object-relational systems typically have application

objects layered on top of underlying relational systems. Some object-relational systems offer search features that find the identities of objects based on their contents and the N-grams within them. This mechanism depends on the object identities captured by the search system and correlated to the objects. While these identities may not be explicitly understood by the underlying SQL database, they are understood by the object-relational system and the search engine layered on top.

Search means finding identities

Search today typically means a system that finds identities of documents, objects, or other things. It is the correlation of the N-grams extracted from these things to the identities that provides search results. Which document identities have the closest match to the set of N-grams submitted with the search?

Naturally, the sorted N-grams are not strongly consistent with the underlying things. There may be things with identities that have not yet been indexed. Sometimes, there are indices that contain the identities for things that no longer exist. *While the things and the indices may slide around, the identities usually stay intact.*

USING IDENTITY TO LEARN

Data science is based on identities, objects, and attributes. It has been used to learn surprising new things. Identities are key to its work.

Data science and observations

Data science revolves around identities. The identities

have attributes. It is the manipulation of these identities and attributes and comparison with other identities that share those attributes that leads to new and deeper understanding.

Identities, objects, and attributes

When observations are made, they are stored as objects and given identities. These objects have attributes. Analyzing the objects may lead to additional attributes being added to them. Continued pattern matching on attributes over large collections of objects can lead to new attributes slapped onto the sides of the objects.

Sometimes, looking at patterns on the objects and their attributes leads to new objects showing the connections between existing objects. This will result in new identities for the new objects. So, the pattern of attributes becomes an identity in its own right, which may lead to new attributes.

Attributes on identities—rinse and repeat

It is the continuous cycle of looking at lots and lots of attributes on the objects and their identities that leads to more attributes. These new attributes are either attached to existing objects or used to generate new objects with their own independent identities.

Data science uses identities to achieve serendipitous learning!

BIG DATA IS LOTS OF IDENTITIES

Big-data systems such as MapReduce,² Apache Hadoop (<http://hadoop.apache.org>), and Apache Spark (<https://spark>.

apache.org] take immutable inputs and apply functional transformations to produce immutable outputs. Because of the immutable nature of the inputs and outputs, it is easy to reason about fault tolerance when pieces of the work fail and are restarted.

Each of these big-data systems leverages the identities of data items to connect work and storage spread across many servers.

MapReduce and Apache Hadoop

These big-data systems look at the data sets they process as a bunch of key/value pairs. Consider MapReduce and Hadoop:

- The map function of MapReduce takes a series of key/value pairs and makes a set of output key/value pairs. These output pairs may be the same as or different than the map function input.
- The reduce function is called once for each unique key and can iterate through the values associated with that key. There may be multiple values for a single key.

Queries, joins, and more with keys

Queries and joins in these big-data environments leverage the keys in the key/value pairs. These are sorted across shards with the map function. The queries and joins are applied by the reduce function handling all key/value pairs with the same key (or identity).

Because the map function can arrange an input key/value into another shaped key/value, MapReduce and Hadoop can query, sort, and join on arbitrary fields in the data. Putting the join fields into the key and sorting allows

for a huge flexibility in function.

Big data means handling lots of keys

Big-data systems require handling *lots* of keys. They can be spread around in a scalable fashion across very large clusters of servers to accomplish massive scale. The identity provided by the keys hooks it all together.

THE “INTERNET OF IDENTITIES”

IoT, or the Internet of things, is the new trend wherein massive numbers of events from disparate devices are processed at high rates.

Internet of things: Identifying the thing

In IoT, an extremely large number of devices that may barely qualify as computers generate massive numbers of events to be processed. Each of these devices will have an identifier in some form. As it generates events, each of these events will have a more detailed identifier that usually specifies its device of origin.

Each of these events will, in turn, have a bunch of attributes that are specific to the device. Events originating from your refrigerator will have different attributes than events originating from your car's transmission or from a security camera at a large stadium.

Querying, joining, and connecting things

Similar to what is seen in big data, each of these IoT events has an identity and a bunch of attributes. These events can be queried, joined, and connected based on their attributes. You can create new events by extracting attributes from a

single event or from a join across multiple events.

An event is an immutable set of attributes with an identity.

THE QUEST FOR IDENTITY

Some of today's most challenging problems come from the quest for identity. Product matching, data science, fraud detection, homeland security, and more all struggle with figuring out when one thing is the same as another thing so identity can be assigned.

Product matching is finding identity

As already discussed, providing an integrated marketplace for stuff sold by wildly disparate merchants is a big challenge. The core of this challenge is matching different SKUs from different merchants with different descriptions to find the same product identity.

This is often made easier with UPC or ISBN codes that actually do match. This leaves the product-matching system with the easier job of comparing attributes to verify identity. Product matching is not always given the boost from shared unique identifiers, and the problem becomes a task of data science.

Data science is finding identity

In data science, there are many objects, each with many attributes. Each object has a unique identity.

- ➡ Attaching new attributes: By comparing many objects and their attributes, the data-science algorithm associates new attributes with existing objects.

- Merging object identities: By examining the attributes bound to sets of objects, the data-science algorithm can realize two objects are one. That, in turn, unites their attributes.

Repeating the attribute/merge pattern causes a new understanding of identity.

Fraud detection is finding identity

Banks issuing credit cards invest heavily in fraud detection, as do retailers and other institutions that accept credit cards. Very large companies that accept credit cards have a strong incentive to detect fraud since their banks will charge them lower fees if their rate of fraud is noticeably lower. Fraud detection is big business.

Fraud detection works by looking at the transactions as objects with associated attributes. Also, credit-card holders are objects with associated attributes. Pattern-matching fraudulent activity from other credit cards to this card can give early warning. Without this matching to find new identities, ecommerce would be very challenging because of the amount of fraud that would get through.

Homeland security is finding identity

Another example of identities and matching comes from looking at patterns of travel, locations, payment types, and more. It is not unusual for an analysis of many travelers to result in similar behavior by ostensibly different people. By realizing they have the same identity, the details known about the different people can be coalesced to gain a better understanding of the risks they may pose.

Laser-sharp vision and blurring details

This coalescing of identities based upon common attributes is the basis for many of the emerging use cases in data science. One perspective is that the set of attributes defines the identity that results from the coalescing. Must the attributes be a match in all their full glory? What makes it OK to have differences? Do we want laser-sharp exactitude in the attribute matching, or is it OK to squint a little bit and blur some details to allow more matches?

Increasingly, the original data (e.g., merchant feeds with product info) is kept and linked to the normalized, matched, and sanitized data. These operations are intrinsically lossy as you strive for commonality with other inputs. Considering the aligned and sanitized common view and comparing it with the individual raw feeds can offer additional insight.

USING IDENTITY FOR ACTIVITIES

Activities are long-running work across time and across computers, and may run across trust boundaries, departments, and companies. An activity is usually handled by having an identifier for the activity and separate identifiers for each step.

Long-running workflow and identifiers

Long-running workflow runs with messages across time and typically waits for external actions to complete. As external events are initiated, somehow an identifier for the event is received when it completes. To deal with an external computer, the identifier is usually tied to outgoing and incoming messages.

Identifiers crossing trust boundaries

Sometimes an activity crosses trust boundaries. Sending messages across companies in a B2B solution opens up trust concerns—perhaps sending messages across departments or even from a Linux box to a Windows box. Each of these solutions offers challenges. The work in these cases is invariably knit together with some form of identity. That identity must have a scope in space that covers all the distrusting participants and a scope in time covering the duration of the work.

It is not uncommon for one system to provide an alias for its identifiers. Messages going out and in are translated between the two identity systems.

Bank check numbers and idempotence

An example of an identifier for long-running work is the check number on the printed checks from your bank. When you make a paper check out to the electric company or the grocery store, the check has a unique identifier. On the bottom of the check are three series of numbers: the ABA (American Banking Association) routing number, account number, and check number. The ABA routing number uniquely identifies your bank. The account number identifies your account within the bank. Finally, the check number is unique within your account.

When your check is handed over to your grocery store, it is deposited in the store's bank, not yours. That bank then records the deposit along with the numbers from your check. The grocery store's bank then forwards the check to your bank, which records the debit and sends money back to the grocery store's bank.

Because of the unique identifier on the check, your bank and the grocery store's bank can implement algorithms to ensure the exactly-once processing of the debit and credit. This has been going on for many years, longer than we've had computers.

Identifiers: The glue that binds and splits

Identifiers are the glue that connects work. It's the ability to connect the work that allows us to split apart our scaling solutions and to connect previously disconnected solutions.

REST: URL-ey binding

REST (representational state transfer³) is an interesting and influential pattern that leverages HTTP and URLs. In the REST pattern, resources are implemented as client-server calls, which are stateless. *Stateless* means that each request from the client holds enough information to process the request at the server without taking advantage of any context stored at the server. The session state is effectively held at the client.

Resources and representations

Within REST, *resources* are any piece of information that can be named. Typically, the name of a resource is a URL.¹ A resource is frequently used to represent groupings of related stuff that may be used to do work. The contents of a resource may be static or dynamic. What is essential is that it can be *named*.

REST resources may project one or more *representations*. Each representation is a view onto the

resource that may or may not be customized for each user. The resource is itself given its own URL(s) as identities.

REST: REpresentational State Transfer

Users wishing to work with the resource are given their own representations as identified with one or more URLs. The resource may have many users. The vast URL identity space is subdivided into representations for each user. Requests for work are accomplished with HTTP PUT commands making modifications to the representation.

Scribbling on the representation

Changing the state projected in the representation is how work is done. The combination of the representation (possibly personalized to the client) and the ability to scribble changes on the representation allows many clients to work with the resource.

URL: Mixing identity, operations, and session state

As changes to the representation occur, responses to the HTTP PUT requests are wrapped up in the URL returned. Contained in that URL is the session state describing ongoing and potentially long-running work for this client.

Identity in the URL captures the operation to be performed.

Identity in the URL captures the session state of the work!

REST: Every verb can be nouned

REST maps a user's perspective to a set of URLs for the

representation. REST also defines the mechanism for invoking computation and work as modifications to the representation. It's *REST* or changes to the representation that cause change.

Every verb (operation) can be nouned (cast as data). These nouns are described as URLs. Operations are cast into identities represented at URLs.

Identity Is RESTing on its laurels

The identity captured in the URL is a large part of why REST is so powerful. The underlying resource has an identity in the URL namespace. Each representation (assigned to a single user) has an identity in the URL namespace. Specific operations are captured, leveraging identity within the URL namespace—a powerful mechanism using the identity of the URL!

SCOPING IDENTITY

Identities must be scoped in space and time so that they don't cause ambiguities. This is, on the one hand, an obvious and silly thing to say. On the other hand, it is a liberating concept.

Identifiers may have permanent unique IDs like those offered by UUIDs. These are powerful and useful. Identifiers may have a centralized or hierarchical authority that assigns their IDs, and that, by itself, offers challenges: Does this authority scale? Is it broad enough in its role to encompass the many different pieces of the solution?

The reality for most systems is that identities span the

participants that see the use of that specific identifier. When merchants interact with a big ecommerce site, they will have shared identifiers for their cooperative work. Still, the merchants may not share the identifiers they use to deal with *private suppliers*. Those private suppliers may have different identifiers used to interact with the manufacturers of their products.

The scope of the identifiers is typically subject to the portion of the workflow that hosts the identifier. There are global IDs like UPC or SSN (Social Security number), but there are also local IDs like SKUs that are defined only for a single merchant.

THE “I”S HAVE IT

Identity is an extremely important part of our systems. Its real power is unleashed when combined with three other “I” words: idempotence, immutability, and interchangeability.

Identity and idempotence

Idempotence is the property that says it’s OK to do work more than once. If it happens *at least once*, the behavior is the same as if it happens *exactly once*.⁵ In general, idempotence is a subjective concept that ignores side effects outside of the plane of abstraction provided by the service.⁶

Idempotence frequently depends on having an identity for the work. In many cases, you need to understand the identity of the operation to decide if you’ve done it before. There are other cases such as reading a record where the work is naturally idempotent because it leaves no effects

when it's performed. In cases where changes are made, tracking that it's already done requires identity of some kind.

Sometimes, the identity used to provide idempotence is a consequence of some connection or session. That works until a new session arrives to retry the failed session.

Managing the requester's identity, the target's identity, and the identity of the work in question are some of the hardest problems in scalable systems that need idempotence.

Banks have used a simple approach to identity and idempotence, with two basic tricks:

- ➔ The transaction's identity is the preassigned check number.
- ➔ The check must typically clear in less than one year after it was written.

The second constraint limits the list of cleared checks the bank must maintain while preserving exactly-once processing.

Identity and immutability

Immutability is the property that something doesn't change. No matter how many times the data is read, the same result is returned. Immutability is the basis for many of today's solutions, from low-level hardware to massively scalable solutions.⁶

Immutability is a relationship between an identity and a result that is unchanging.

Without some formalized notion of identity, you don't have immutability.

We may see immutable identifiers for changing stuff. A product ID may be fixed for years while its description evolves.

Identity and interchangeability

Interchangeability can be viewed as a duality with immutability. Rather than asking, "Is this thing identical?" to what we had before, we ask, "Is this thing equivalent?" to what we had before. Is it good enough?

When manufactured items are all brand new and identical, you can be happy taking any one of them from the warehouse, assuming they're not damaged. There is an identity for the product, and that identity means any one of them will do. They are interchangeable.

When reserving a room at a hotel, you accept that one king-sized nonsmoking room is as good as another—even if one is next to the elevator and really noisy. The group of rooms labeled as king-sized nonsmoking is considered equivalent, and *there is an identity for any one of those rooms*. You reserve one from the pool of rooms without knowing exactly which one.

Recall that an identifier for a product description in a product catalog refers to an ambiguous version of the product description. That's OK... any one will do, as the versions are interchangeable.

Identities may refer to an abstract grouping of equivalent things.

The judicious use of ambiguity and interchangeability lubricates distributed, long-running, scalable, and heterogeneous systems.

The real art of interchangeability lies in finding a way to identify the equivalent set of individuals...

Where they must be the same and where differences are acceptable.

CONCLUSION

It used to be that we focused on one application running on one computer accessing one SQL database. While we

may have had application-based identifiers (e.g., Social Security numbers), the underlying system was based on values in cells.

Relational algebra related values to other values.

As systems *cleave apart* for scale, *cleave apart* to provide management or trust boundaries, or *cleave together* to integrate solutions, identifiers and identity form the glue that binds solutions. Identities also formalize the separation of disparate and distrusting solutions. Cleaving apart or cleaving together requires identities.

When we bind work together with identities, the interesting tension is, “What constitutes

Related articles

➡ Pervasive, Dynamic Authentication of Physical Items

The use of silicon PUF circuits

Meng-Day Yu and Srinivas Devadas

<https://queue.acm.org/detail.cfm?id=3047967>

➡ How Do I Model State? Let Me Count the Ways

A study of the technology and sociology of Web services specifications

Ian Foster, et al.

<https://queue.acm.org/detail.cfm?id=1516638>

➡ How to De-identify Your Data

Balancing statistical accuracy and subject privacy in large social-science data sets

Olivia Angiuli, Joe Blitzstein, and Jim Waldo

<https://queue.acm.org/detail.cfm?id=2838930>

the identity?” What precisely is identified by a king-sized nonsmoking room? Where did we deliver the message that was guaranteed to be delivered?

New emerging systems and protocols both tighten and loosen our notions of identity, and that’s good! They make it easier to get stuff done. REST, IoT, big data, and machine learning all revolve around notions of identity that are deliberately kept flexible and sometimes ambiguous. Notions of identity underlie our basic mechanisms of distributed systems, including interchangeability, idempotence, and immutability.

Finally, don’t be too picky about calling this identity. We see identity as names, keys, pointers, handles, IDs, numbers, identifiers, UUIDs, GUIDs, document IDs, UPCs, ASINs, employee numbers, file names, Social Security numbers, and much more.

Truly, identity by any other name *does* smell as sweet...

References

1. Berners-Lee, T., Masinter, L., McCahill, M. 1994. Universal Resource Locator. Technical Report, Internet Engineering Task Force, Draft RFC; <https://dl.acm.org/citation.cfm?id=RFC1738>.
2. Dean, J., Ghemawat, S. 2004. MapReduce: simplified data processing on large clusters. *Proceedings of the Sixth Symposium on Operating Systems Design and Implementation* 6, 10; <https://dl.acm.org/citation.cfm?id=1251264>.
3. Fielding, R. 2000. Architectural styles and the design of network-based software. Ph.D. dissertation. University of California, Irvine.

4. Helland, P. 2016. The power of babble. *Communications of the ACM* 59(11), 40-43; <https://dl.acm.org/citation.cfm?id=2980932>.
5. Helland, P. 2012. Idempotence is not a medical condition. *acmqueue* 10(4), 30; <https://dl.acm.org/citation.cfm?id=2187821>.
6. Helland, P. 2016. Immutability changes everything. *acmqueue* 13(9); <https://queue.acm.org/detail.cfm?id=2884038>. (First printed in the Biennial Seventh Conference on Innovative Database Research (January 2015)).
7. Helland, P. 2005. Data on the outside versus data on the inside. In *Proceedings of the Conference on Innovative Database Research*; <http://cidrdb.org/cidr2005/papers/P12.pdf>.
8. Helland, P. 2017. Side effects, front and center! *Communications of the ACM* 60(7), 36-39; <https://dl.acm.org/citation.cfm?id=3080010>.

Pat Helland has been implementing transaction systems, databases, application platforms, distributed systems, fault-tolerant systems, and messaging systems since 1978. For recreation, he occasionally writes technical papers. He currently works at Salesforce.

Copyright © 2018 held by owner/author. Publication rights licensed to ACM.