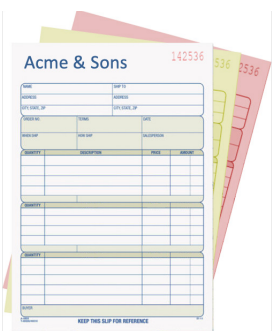
Autonomous Computing

PAT HELLAND

WE FREQUENTLY COMPUTE ACROSS AUTONOMOUS BOUNDARIES BUT THE IMPLICATIONS OF THE PATTERNS TO ENSURE INDEPENDENCE ARE RARELY DISCUSSED.



Starting 23 years ago, I spoke about how autonomy impacts our designs. By 2001, I had presented my ideas at a number of conferences. I regret not having set them to prose at the time. Many of the ideas moved forward to be called SOA (service-oriented architecture) in the early 2000s. Now, in 2022, as these principles still influence our designs, we don't talk enough about them or their implications.

This article introduces a concept whimsically called a fiefdom, a collection of autonomous computing and data designed to work with independent outsiders. This is not the same as Byzantine consensus since we don't really draw consensus across these boundaries. Fiefdoms are autonomously controlled and managed. Since they are autonomous, they won't do work in shared transactions. Rather, they count on the flow of messages across autonomous boundaries, the data contained in those messages, and still accomplish business while preserving their independence.

A related component in the emissary, another whimsically named role for code that is designed to work

with a fiefdom across its autonomous boundary while being an outsider. It runs outside the autonomous fiefdom. The emissary makes working with the fiefdom simpler by providing an easy-to-use interface running close to the external partner. Since messages generated by the emissary match those expected by the fiefdom, incoming work usually succeeds, and more business is accomplished.

If you are autonomous from your business partners, you won't share transactions with them. Transactions aren't used across organizations since this may lock up YOUR database if the OTHER organization messes up. Without transactions, you must use multiple messages over time. But running across time to get a long-running business task accomplished means you must deal with uncertainty about the outcome. This, in turn, implies you must reconcile the work's outcome. All of this is implied by autonomy.

Working across boundaries happens over time by correlating related messages. Before computers, this was done with paper forms having unique serial numbers, multiple parts to each form, and successive modifications over time. This article introduces an abstraction called a collaboration to explain how this works. Like fiefdoms and emissaries, this is a pattern with many possible implementations.

These patterns are rooted in centuries of business practices helping companies work across boundaries. While all the benefits I discussed long ago still apply, there are even more ways to leverage these patterns in this distributed and replicated world. This article is finally the prose version of my many presentations on autonomous computing.

When originally discussed more than 20 years ago, fiefdoms were conceived of as a single database surrounded and protected by a two-tier application. As time has moved on, there are examples of fiefdoms running over hundreds of thousands of computers providing massive scale. The autonomous computing pattern is agnostic to scale both large and small. Rather, it focuses on the importance of working across multiple interactions and correlating later messages to earlier ones using multiple related messages within a collaboration. Collaborations encompass business-specific sequences of messages, how they interact, and how they are resolved.

This article introduces fiefdoms, collaborations, and emissaries and shows examples within computing and within our daily lives. It examines how emissaries work outside the autonomous boundary and are convenient while remaining an outsider. The article examines how work *across different fiefdoms* can be initiated, run for long periods of time, and eventually completed.

It concludes by reiterating that autonomy and shared work are based on both collaborations and the data shared between partners.

FIEFDOMS, EMISSARIES, AND COLLABORATIONS

Let's introduce a few concepts seen in the common pattern referred to here as autonomous computing.

Fiefdom is an English word for an estate with its own independent leadership, rules, and sphere of control. Historically this included duchies, baronies, or other independent legal domains. Each fiefdom is largely independent of other places.

An emissary is a person sent on a special mission as a diplomatic representative. Unlike an ambassador, an emissary has NO authority but can only play nicely, suck up, and try to ease the relationship.

A collaboration supports messaging over time. Like the paper forms that empowered business before computers, a collaboration is modified by both emissaries and fiefdoms as they send messages back and forth. In addition to tying together related messages, a collaboration provides a means to connect the work within a fiefdom or emissary to the long-running internal state supporting the cooperative business work.

I am describing computing patterns in many existing applications. These patterns may be supported by relational data, no SQL key values, and many other techniques. Correlating related messages to the work they stimulate empowers cooperative business work *even when the partners are autonomous from each other*.

Let's introduce the notion of a fiefdom within the autonomous computing pattern. Then, we'll consider the special role of an emissary: making it easier to work with the emissary's fiefdom. The collaboration messaging pattern gets our attention next. Each fiefdom's independence changes the nature of data both inside and outside its rigid and autonomous boundary.

Fiefdoms and autonomous computing

The only way to get work done with a fiefdom is by sending a message. Each fiefdom has messages it accepts and

specific requirements of partners sending work.

When I go to my bank's ATM to withdraw cash, it lets me do only a few tasks. I can withdraw cash, deposit a check, transfer funds, and check my balance. Any attempt to get a JDBC (Java Database Connectivity) connection to the bank's back-end database fails. It's like my bank doesn't trust me! How do parties work together when one is independent from the other?

Autonomous fiefdoms can be computing software and hardware, or it can be a bunch of people trying to do business together without computers.

REQUESTING SERVICE FROM A FIEFDOM

Fiefdoms define their own rules for doing work. If you want cash, you must insert your ATM card and provide your PIN. If you want to sell your products on the shelf of my megastore, you must send me electronic messages describing the products and beg my permission. In general, the shape and description of the interaction are defined by the fiefdom.

Sometimes, multiple fiefdoms work together. Overall, the fiefdom with the most economic sway in the relationship defines the protocol and messages to be used. The economic dog wags the economic tail.

PRIVATE DATA WITHIN FIEFDOMS

Encapsulated within the fiefdom is private data. This is internal and rarely shared with outsiders. My bank knows my accounts and their balances. It doesn't give ME this information about YOUR accounts. Heck, it doesn't tell ME about MY bank accounts any more than necessary.

Private data encapsulated within the fiefdom may take many forms. It contains the fiefdom's most precious thoughts and business knowledge. It is internal and should remain internal.

TRANSACTIONS AND FIEFDOMS

Transactions across boundaries imply intimacy and trust across boundaries. Classic database transactions require coordination and may lock up data records awaiting the transaction's outcome. As a autonomous fiefdom, no way am I going to lock up MY database records awaiting YOU.

If we're autonomous, we have no distributed transactions. Each fiefdom uses database transactions INSIDE its belly. There is no way it is going to share a transaction with some miscreant business partner knocking at the gate of the castle.

A fiefdom is a design pattern for long-running interactions *across autonomous boundaries*. Work happens with a sequence of related messages over time to cooperatively perform work. This is how it was done centuries ago and how it's still done today.

Collaborations for long-running work

A collaboration is an abstraction for a set of messages into and out of a fiefdom for a single long-running business operation. This is not a new idea.

When I was a kid in the 1960s, there were no computers in our daily lives. Sure, large companies had them in the back rooms, and the government and military had big

mainframes, but we had no interaction with computers day to day. Instead, we had paper. When shopping, going to the doctor's office, or getting something repaired, you filled out forms that controlled the activity. I vividly remember the multipart, multicolored forms required to get something done.

Sometimes, I would accompany my mother or father on some errand—dropping off dry cleaning, getting the car repaired, or ordering something at the department store. A specialized form was pulled from under the counter and my parent would fill out part 1 on the front page, being careful to push the pen hard enough so the writing appeared on the last page.

Each form was preprinted with a serial number in the upper right corner. When my parent finished filling out part 1, the clerk would tear off the back page of the multicolored form and hand it back. The rest of the pages were kept at the store or repair shop for internal use. Different pages (with their unique colors) were kept within the departments, so every participant had a record. These forms knit together the work of the internal departments of the business and tied it back to the customer.

The second page from the back was for the front desk. It was torn off and filed by serial number in a file cabinet, and the rest of the form went into an outbox to be routed to an internal department within the business. They did some work, filled out part 2, tore off the back page, and sent the form on its journey. The front desk's file folder was organized by the form's serial number. When work was completed and my parent picked up the purchase, dry cleaning, or repaired item, the paper form was filed by its

serial number in the "completed work" file cabinet.

Each multipart paper form is a collaboration. It captures the request, tracks ongoing work, and ensures everything is done before the paper form is retired into the "completed work" file folder.

Doing work across autonomous boundaries frequently requires multiple interactions. Correlating these interactions used to require paper forms. In computing interactions, multiple messages must also be correlated and tied to earlier or later work. This is the collaboration pattern. Collaborations are implemented within most applications by tracking a unique number within each message. This is used to find the related messages in the collaboration and move the long-running work forward. This is how the abstraction becomes reality today.

USING REFERENCE DATA TO FILL OUT FIELDS IN A COLLABORATION

One of the highlights of my mom's year was the twice-yearly Sears catalog. This was a massive book with hundreds of pages of stuff available for sale. Dresses, kitchen items, tools... you name it, and it was in the catalog.

When the catalog arrived, many of us in the household would study the wondrous array of possible purchases. Included was a form to fill out with item numbers and prices. These were diligently copied into the form to enumerate the desired stuff. Mom would enclose the form in an envelope and send it to Sears along with a check

for payment. Weeks later, it was like Christmas when a delivery truck brought you all these things that were not easily attainable in your hometown.

The information in the Sears catalog was, in my nomenclature, reference data. Its sole purpose was to help you fill out the order form. Since it came only twice a year, it was somewhat stale with passing time. The catalog clearly stated how long the item numbers and prices were valid.

We used reference data to fill out a collaboration to do work across autonomous boundaries. In this example, Sears mail-order business was a fiefdom. We had no idea which internal departments within Sears were involved in fulfilling the order. Both the reference data and the collaboration had a bounded lifetime. They were created, spread across the autonomous boundaries, used for a while, and then retired. Eventually, an expired Sears catalog became a doorstop or got fed into the fireplace.

Emissaries: Helping the interaction with fiefdoms

I purchased my first home in 1979, a tiny, crappy place, too small for a family of six. To get the mortgage, we went to a mortgage broker. People in this profession are paid to be nice and make it easier for the borrower.

The mortgage broker was NOT employed by any bank. Typically, these brokers would help the buyer get a loan from one of many banks. They had information about various banks, including rate sheets, qualifications, and lots of stuff barely understandable to a 23-year-old first-time home buyer.

When a bank was selected, we filled out lots of forms

using the reference data cached within booklets in the mortgage broker's desk. Upon receiving all the data, the bank did NOT trust it and ordered a credit rating and employment verification. Everything in the forms was checked by the autonomous bank, which found this convenient since the mortgage broker knew how to fill out the forms correctly, making bank's job easier.

Mortgage brokers are real-life emissaries. They are independent from the bank but do a heck of a lot of useful work. Unlike some emissaries, a mortgage broker can be the front end to many different banks.

Within computing systems, there are many uses of the emissary design pattern. Shopping at large ecommerce sites is done using an emissary as the front end. In this case, the emissary is implemented with thousands or tens of thousands of servers supporting shopping carts, product catalogs, images of products, recommendations, reviews, and much more. Everything before the shopper pushes Submit is an emissary pattern. After you push Submit, the request is sent to the back-end system for payment processing, inventory check, shipment scheduling, and more.

Complex and distributed back-end processing implements the ecommerce fiefdom. Indeed, there will likely be many internal fiefdoms inside the big ecommerce back end.

SOMETIMES MANY EMISSARIES; SOMETIMES MANY FIEFDOMS

Consider your mobile phone and its applications. Email comprises both a back-end fiefdom implemented by your

mail server(s) and an emissary on your mobile phone. I have my phone's email connected to my work email, as well as two personal accounts. Browser-based mail clients may interact with the same mail servers or a different overlapping set. The protocol between each emissary and its back-end fiefdoms is a fancy collaboration for each of the emissary-to-fiefdom pairings.

This may seem like a shared database with optimistic concurrency, but it's not. It is a protocol to add information, including a possible tombstone denoting the intention to delete a mail message. Mail messages are never really deleted. If you doubt that, ask any FBI agent. Instead, they are archived with tombstones indicating they're not to be shown.

Emissaries are NOT trusted by fiefdoms. They are outsiders. Everything they do is verified behind the castle wall and inside the autonomous computing environment. Emissaries may provide convenient front ends for many different fiefdoms, unifying how the user sees them behind a single mortgage broker or a single mail client front end.

EMISSARIES AND THEIR DATA

Emissaries may have reference data, perhaps in bulk form like the Sears catalog or in offline copies of stale email messages. In addition, an emissary may have a significant per-user state capturing your abilities as a borrower, your shopping cart containing proposed purchases, or your view of the downloaded email messages on your phone.

Emissaries are a design pattern that makes working with fiefdoms easier. They execute outside the autonomous fiefdom and specialize in simplifying the multimessage collaboration with the fiefdom.

The key to the pattern comes from how data is used within the multimessage collaboration between the emissary and the fiefdom it is designed to support.

What Is offline processing?

On one hand, offline processing is pretty easy. Just buffer up enough reference data, take it offline, add messages to the shared collaboration, and squirt it over to the fiefdom when you reconnect. Sitting at my mother's kitchen table filling out forms for the Sears catalog is offline processing. Working with mortgage brokers at their office is offline processing. So is reading and writing email using my mobile phone.

Of course, some applications have additional challenges. Let's address some of the easier problems first. How can you know what data to bring along? Of course, that's both application (or domain) specific and may depend on the user's preferences or history using the application.

RECURRENT REFERENCE DATA

Recurrent reference data is something you sign up for on a regular (or irregular) basis. My mom's biannual Sears catalog was an example of offline recurrent reference data.

Email is also offline recurrent reference data. Many of

us have multiple email accounts held with different mail providers. In each case, an email account has a notion of the messages in the account. A browser may access my email via one mail provider in a synchronous fashion. The mail client on my laptop, tablet, and/or mobile phone caches the offline state of my messages. Each emissary (my device's mail client) collaborates with its fiefdoms (one of the mail servers) whenever it can. The intermittently offline emissary (my mail client) adds information to the collaboration shared by this mail client and a specific mail server.

This pattern of recurrent reference data is increasingly common as the complexity of work increases—for example, the many different shared folders, channels, groups, and team-blather sessions that facilitate teamwork. Application-to-application usage of recurrent reference data is nascent and emerging.

USING COLLABORATIONS FOR OFFLINE WORK

Like a paper form, a collaboration provides limited communication across autonomous boundaries. It may be implemented with some buffering within the emissary. Implementing buffering is usually the responsibility of an application developer. The autonomous computing pattern is usually handcrafted within each application. Allowing asynchronous delivery of the messages from the emissary to the fiefdom supports one piece of the puzzle needed for offline work. Unfortunately, many of these mechanisms are reimplemented over and over for different applications.

Offline processing is simply asynchronous and buffered changes to a collaboration shared with the fiefdom. Of course, some fiefdoms must also reconcile concurrent

requests from many offline emissaries working with the fiefdom over multiple collaborations. (Some of these patterns are discussed in a longer version of this paper at https://arxiv.org/pdf/xxxx.xxxxx.pdf.)

COMPLEX LONG-RUNNING BUSINESS WORK

Collaborations provide ordered full-duplex messaging *within a single collaboration*. Across separate collaborations, all bets are off and no guarantees are made. Within a collaboration, there may be ongoing work for minutes, hours, days, or months. This implies some form of durable storage on each end, within both the fiefdom and the emissary. Many collaborations are ongoing at the same time. Each domain-specific application defines *its notion of a single ongoing piece of work* and leverages the collaboration's ordered full-duplex set of messages to get the job done.

When you can't do distributed transactions, you need to manage related messages. Collaborations simply group related messages for a single job over time. How you do this within the pattern will vary with different implementations.

Richer partnerships for long-running work require richer message patterns. Negotiating for the delivery of thousands of parts in different shipments scheduled to span months is complex. Many messages may be required before any shipment happens, during the timeframe of

delivery, as well as to resolve the payment schedule and the closure of the work. These are typically correlated with an identifier strongly analogous to the serial number in the upper right-hand corner of the paper forms from my misspent youth.

Within the collaboration and its many messages, you may see messages from the participating fiefdoms or emissaries written into different "Parts" of the collaboration. These are simply classes of messages sent by a collaborator for a specific purpose. These may flow in a logically full-duplex fashion with each collaborator loosely coupled with the other.

All collaboration-based work is offline! Collaborators (fiefdoms and/or emissaries) don't share transactions; they are ALWAYS offline. Semantically, the work is not atomic and must ratchet its way forward step by step to complete the task.

Rethinking data across trust boundaries

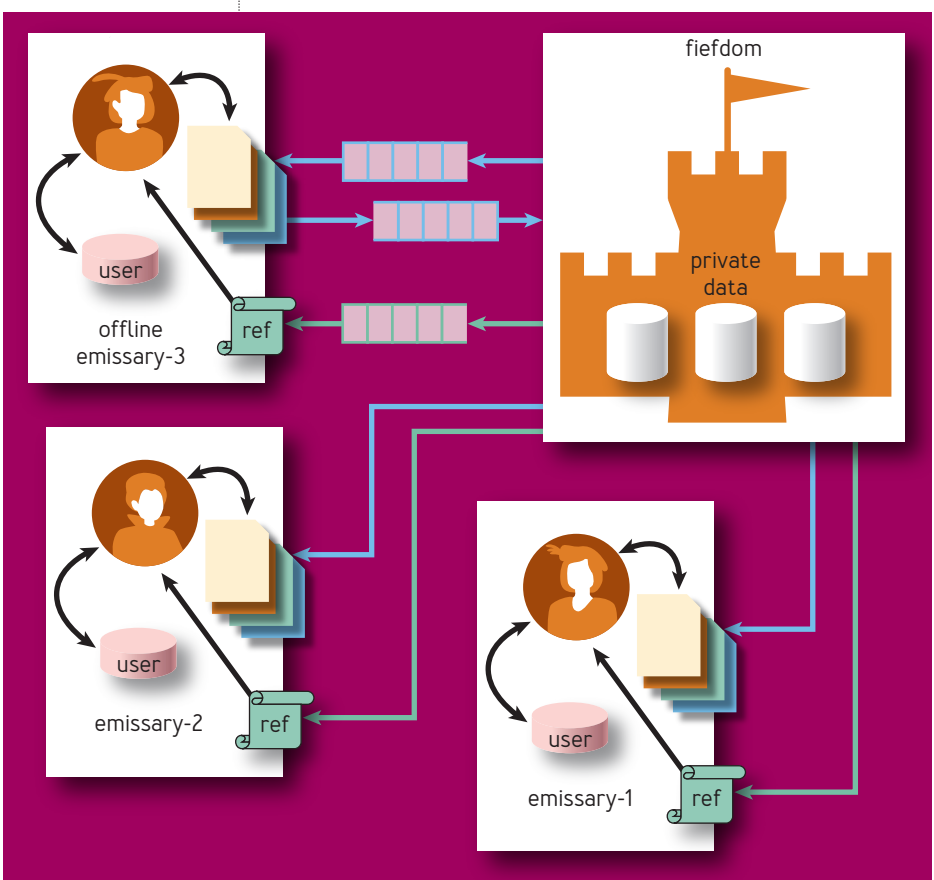
Data has certain roles in this design pattern, illustrated in figure 1:

- **Private data.** Inside the fiefdom.
- **Single user.** Emissary data.
- **Reference data.** Stale immutable data.
- **Collaborations.** For interaction between fiefdoms and emissaries.

There are special roles for data in the netherworld between these collaborators. Like all data flowing outside

an autonomous boundary, this data is identified, possibly versioned, immutable (per version), and has descriptive metadata bound to it when created. This is outside data as described in my 2005 paper "Data on the Outside

FIGURE 1: **DATA ACROSS TRUST BOUNDARIES**



versus Data on the Inside" (Conference on Innovative Data Systems Research).

In each case, data outside the fiefdom uses identifiers to knit it together:

- Collaborations have their messages associated by an identifier (e.g., an order-id).
- Reference data will likely have products, prices, and options tied by one or more identifiers. Email is knit together using internal unique message-ids. In most cases, this is versioned over time.
- Single-user data may correlate the ongoing business work to both the active collaboration and the reference data. Shopping carts are single-user data.

Inside the fiefdom, identifiers tie the collaborations and ongoing work with internal state.

Both inside and outside the fiefdom, these identifiers may be kept in a relational database or may be kept in some other durable storage. They must be durable to ensure they are ready for the next interaction with a related collaboration. See my 2019 article, "Identity by Any Other Name" (*acmqueue* 16(6)).

WORKING ACROSS FIEFDOM BOUNDARIES

Let's look at working across autonomous fiefdoms. Collaborations are used to cooperatively complete work while ensuring independence from partners outside the fiefdom. The fiefdom accepts uncertainty in its plans and engages in an ongoing dance of accepting uncertainty and risk and resolving that uncertainty and risk.

Working without shared transactions

You can't count on distributed transactions across boundaries. Transactions may happen on either side of a collaboration but may not span boundaries. *If you are autonomous, you don't have shared transactions!*

The alternative to a shared transaction is a shared sequence of messages. That is the collaboration defined here. The bidirectional sequence of messages is correlated and understood by both collaborators to be related as work proceeds over time.

This implies identity for each of the two collaborators, as well as their ongoing collaborations. These identities must be long-lived to continue the work. In real-life business interactions, there's also some mechanism for ensuring payments by a collaborator. For large businesses working together, we may see a contract with a threat of legal enforcement. For B2C (business-to-consumer) work, the customer's credit card ensures payment to the business. The customer may challenge any wayward credit card charges. This gives them independence from the fiefdom while accomplishing business.

Each interaction in a collaboration starts by digging deeper into the solution and completes by digging back out. When shopping at a large ecommerce site, you browse, check reviews, look at similar products, and add items to your shopping cart. The retailer's emissary enhances the experience. The quality of this offline (or pseudo-offline) experience contributes a LOT to the retailer's profitability.

The emissary for ecommerce shopping is effectively offline because it's not sharing a transaction. You may see: "USUALLY SHIPS IN 24 HOURS," which tells you *nothing at*

all but is incredibly useful. It is a probabilistic statement with no transactional guarantees. Still, based on this phrase, I'm always adding or removing items from the cart as I shop for the grandkids. It's useful!

When the shopping cart is full and the user pushes Submit, lots of steps begin. Usually, the back-end work is asynchronous over seconds, minutes, or days. Here are some typical steps:

1. *Emissary starts the collaboration.* The front-end emissary combines the shopping cart, user information (including addresses and credit cards), product-ids, pricing information, gift wrapping requests, and more into a single message. This message is enqueued to the back-end system at the ecommerce retailer for order processing.

2. *Async but frequently fast.* Normally, you hear a "bleep" as an email comes to the human user seconds after the Submit. Sometimes, this "bleep" happens 30 minutes or so later if the back-end order-processing system is busy, and its work queue backs up.

3. *The collaboration begins.* This starts a collaboration to implement the order. The order-id correlates all work for this ongoing request. Payment is extracted from the credit card. Messages, sometimes via email, advise the consumer of shipment, delays in shipment, out-of-stock problems, and completion of the order.

4. *The collaboration completes.* When the product is delivered and the credit card charged, that usually completes the collaboration. In reality, it is kept alive for a while in case the customer cancels the order. Only after 30 to 90 days will the collaboration be complete.

In this example, email messages to the human fulfill some of the collaboration's role. The emissary's reification is a bit fuzzy. Still, the pattern is identical for many business operations.

Collaborations involve tentative work, their possible cancellation, and their hopeful completion over time. Cooperative work by autonomous parties involves multiple steps within a framework for financial damage to the other party, if necessary. *Correlating related messages is essential.* The use of an order-id or some other identifier to knit together the collaboration is an example of the general concept of using identifiers to connect work in distributed environments.

Collaborations, uncertainty, and reconciliation

It is common for autonomous fiefdoms to accept and resolve business commitments. The nature of these varies based on the domain of the business, their protocols, and their appetite for risk. Some enterprises go out on the limb to get business, accepting occasional loss as a cost of business. Others want strong assurances to move forward.

In all cases, sequences of correlated messages flow between prospective collaborators.

It is the nature of these correlated messages, their identity as a set of messages, and the patterns of their use that shape the autonomous computing pattern.

Using a sequence of messages, the communicating fiefdoms (or trust boundaries) establish some form of limited trust, usually by ensuring financial compensation should things go wrong. Time passes and messages flow to deepen the relationship, accomplish the goals, and back their way out of this single collaboration. Commitments may fail and the businesses recover.

It is the correlated sequence of messages called a collaboration that empowers this work across fiefdoms, both B2B (business-to-business) and B2C (business-to-consumer).

Inside a fiefdom, the business manages its risk, balancing between commitments made via its collaborations and its challenges meeting those commitments.

Work using collaborations is contractual and may involve financial penalty. The fiefdom sets rules and regulations for cooperating over the collaboration, including various contingencies. This sequence of events takes time (perhaps months or longer) and time-based exceptions are the norm. See my 2009 paper "Building on Quicksand" (Conference on Innovative Data Systems Research) for a discussion of overbooking, overprovisioning, and coping with uncertainty in long-running autonomous computing.

WORKING WITHIN A FIEFDOM

How do you design an autonomous fiefdom? How can it support the long-running work necessary to meet its collaborative commitments over time?

Typically, the work is broken down into a couple of different roles within the fiefdom:

- Activities. Data and computation used to track the fiefdom's work, or part of the work, for a single collaboration.
- Resources. Code and data used to manage shared items coordinated across multiple activities such as widgets in inventory or space on a truck needed for a shipment.

These patterns support external collaborations and are frequently composed using many internal collaborations.

Inside a fiefdom: activities and activity data

When an incoming collaboration arrives at a fiefdom, an internal data structure is allocated to track the collaboration's messages, work stimulated by these messages, and the completion of the work. These activity data structures are encapsulated by fiefdom-specific activity code.

Work advancing this activity is stimulated by any of the following:

- An external message arriving on an external collaboration.
- An internal message on an internal collaboration used to drive intra-company work.
- A timer advancing the activity to a new state and perhaps taking new actions.

Fiefdoms create mechanisms to manage their activities'

workflows. There are many ways to make this pattern work. Here we focus on the pattern and its implications.

Activities do not live forever. Their life cycles are driven by the workflow needed to complete business with the collaboration's partner. They are created in response to collaboration messages, live for a while (perhaps seconds, hours, or months), are retired into a read-only state, archived, and deleted years later.

Inside a fiefdom: Resources and resource data

Tangible things are managed using resources such as inventory, shipments on a truck, passengers booked for a single flight, king-size nonsmoking rooms booked for a particular night, billable dollars per customer for a single month. These are examples of resources.

Frequently, a resource and its surrounding resource manager must deal with uncertainty. Resources may be allocated for a customer but need payment before shipping. Scheduled space on a truck goes unused if an incoming truck is delayed. A resource manager for outgoing shipments may place a waiting box on an outgoing truck but only if the activity controlling the waiting box confirms the change.

Resource managers work with time, incoming resources, outgoing resources, overprovisioning (to ensure customer happiness), or overbooking (to manage expenses). They cope with an ever-changing world across many competing activities. Sometimes they fail to meet obligations to one activity because of the demands of another activity.

Inside each resource manager are little activities that track the single allocated resource (or pending resources) and correlate back to a larger-scale activity tied to an external collaboration. These, in turn, impact the allocation of the precious resources for other activities.

Resources see increasing and decreasing uncertainty, especially when overbooking their managed resource. Only when a flight flies away do you really know who's on it.

Many collaborations and their messages connect these many activities and resources. This can provide a unified way to wake up activities, resources, and collaborations in response to incoming events or timers.

AUTONOMY, COLLABORATIONS, AND DATA

Autonomous computing is a pattern for business work using collaborations to connect fiefdoms and their emissaries. This pattern, based on paper forms, has been used for centuries.

How can we make it easier for people to follow this pattern as they solve business problems with computers? We should help them focus more on their business and less on hooking stuff together!

A longer version of this paper is at https://arxiv.org/pdf/xxxx.xxxxx.pdf. It has more examples and discussions of scale, replication, and life cycles of fiefdoms, emissaries, and collaborations.

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 works at Salesforce. His blog is at pathelland.substack.com.*