Engagements: Building Eventually ACiD Business Transactions

Pat Helland
Salesforce.com
1 Market Street
San Francisco, CA 94105
1-415-546-5881
PHelland@Salesforce.com

Don Haderle
Gainfully Retired
and Having Fun
DonHaderle@yahoo.com

ABSTRACT

Business Transactions describe long-running operations that may span many discrete systems. This has been an area of research for many years [6],[15]. While these operations involve tasks that may or may not be transactional in the traditional sense, the business still wants many of the same Atomic, Consistent, Isolated, and Durable characteristics that we know and love. Providing ACID can be difficult in an environment that includes:

- Long-Running Operations: The work may span weeks or days.
- Autonomous Participants: Independent systems may renege on promises or simply lose their state.
- Schismatic Participants: Sometimes replicas of participants may independently do the work more than one time.

We propose a new communication and storage mechanism called *engagements*. Engagements connect the participants of an Eventually ACiD Transaction. Participants are autonomous and may break their commitments. They may also be schismatic and composed of replicas that take independent and redundant actions. Engagements offer support for coping with the problems that may arise. This allows a new perspective on ACID:

- **Eventual Atomicity:** Schismatic participants may do the work two or more times. With the help of engagements, this eventually becomes one operation with the redundant ones canceled. Autonomous participants may refuse to do what's been promised. Engagements ensure these operations are eventually performed or the breakage is escalated for help.
- **Eventual Consistency:** The business transaction is eventually completed. Furthermore, its outcome allows reordering of the schismatic or alternative redundant work while achieving the same outcome across all the replicas. The business transaction will eventually become consistent or an escalation happens.
- **Probabilistic Isolation:** The promise protocol knits together the business transaction allowing for far greater commutativity. Work may ignore side effects at a lower layer of abstraction (using open-nested transactions). Hence, lower level concerns are semantically isolated from the business transaction. Still, while schismatic and autonomous work is eventually cleaned up, concurrent work may possibly see through the isolation. Concurrent work may see dirty or stale data from other business transactions and we don't deal with this problem.
- **Eventual Durability:** Engagements capture the promises made by participants as they interact with the rest of the business transaction. They also allow the participants to record their state as they chatter. A less-than-durable participant can be resurrected and fulfill his job when presented with his state from an ongoing engagement. Eventually, the collection of participants can repair the temporary loss of some participants.

The views in this paper are those of the authors. They may not reflect the views of any of their employers, real or imagined.

6th Biennial Conference on Innovative Data Systems Research (CIDR '13) January 6-9, 2013, Asilomar, CA, USA.

1. INTRODUCTION

Eventually ACiD Business Transactions are a pattern of long running work comprising many individual "classic" transactions spread across many disparate systems. These systems are independent (autonomous) and are frequently implemented as a set of replicas that will sometimes give confusing answers.

We will propose a messaging and state management abstraction called *engagements* which we argue can lower the challenges faced in this somewhat chaotic life in the real world.

In this introduction, we first cover the key insights that drive our proposals. We describe our assumptions about the unreliability and idiosyncrasies of the participants comprising long-running business transactions. We talk briefly about engagements and their implementation (with more to come later in the paper) and then introduce the notion of speculative execution across replicas. We describe how an application program is written in this world. Next, we mention the mechanisms that allow engagements to cope when a participant simply refuses to keep their earlier commitments. Finally, we sketch the rest of the paper.

1.1 Key Insights

This paper is based on a number of key insights:

- **Business Resources Appear Side-Effect-less:** As businesses manage their resources (e.g. hotel rooms, plane flights, or delivery of widgets), they make commitments based on the individual resources. Sometimes, the business itself does work spanning resources but this is hidden from the higher-level business transaction receiving the commitment.
- **Business Resources Are Leaves of the Tree:** The transaction forms a tree. The leaves of the tree can access shared resources. The interior nodes are not allowed to directly access resources from outside the transaction.
- **"Schismatic Versions" Isn't the Same as Different Replicas:** There may be schismatic behavior at first when the replicas are acting independently. Soon, the replicas hear the news about their siblings' different actions. All the replicas will retain all the possible courses of action until a winner is selected. What's important is a predictable winner-selection algorithm that is uniformly applied by the replicas.

- **Schismatic Versions Are OK as Long as No One Notices:** A collection of schismatic participants can get into a complex world of possible answers and no one will mind as long as this settles into a single "truth" by the time anyone notices.

1.2 Business Work when Things Go Wrong

In our experience, there are a couple of ways in which participants in long-running business transactions can be a pain-in-the-butt:

- **Autonomy:** The participant in the long-running transaction may renege when it's time to finish the work. This may be due to a lack of durability, overbooking, or some concurrent problem from related transactions (e.g. a deposit bounced).
- **Schisms:** Sometimes business operations have replicas that do the work extra times. This may be due to replication within or across datacenters, multiple team members doing a workflow step, or as a person having multiple input devices.

Rather than try to harden systems to avoid these issues (which you can't really avoid, anyway), we're exploring how to make resilient business transactions *while embracing these challenges*. We think this may someday provide more stable solutions.

1.3 Engagements: Tracking Messages & State

We are proposing a new (and non-existent) piece of plumbing called an *engagement manager* that will implement a new programming construct called *engagements*.

Engagements combine both communication and storage as they connect the programmatic participants of a long-running business transaction. Each engagement connects two possibly replicated participants in the business transaction. As we shall see, the participants in a business transaction are always arranged as a tree with the initiating participant as the root of the tree.

Engagements have special mechanisms to support the business transaction when its participants are schismatic or autonomous.

Engagements provide messaging. Each replica of a participant processes an incoming message as a classic database transaction, modifies its internal state, and optionally sends out messages on its engagements. This happens separately for each replica.

1.4 Speculative Execution

Sometimes, replicas wait passively while one sibling processes messages. Sometimes, sibling replicas of participants execute in parallel. Most of the time, this doesn't matter as the replicas typically do the same thing and the redundant (but identical) work may be collapsed. Sometimes, however, the replicas set out in their own direction and perform completely different work. This occurs when either the resources accessed return different results or racing messages in the tree arrive in different order.

Arguably, this is similar to speculative execution [18] inside processor chips or other environments. The trick is to ensure this speculative work is cleaned up to result in a single acceptable answer and that any visible side effects are removed.

Engagements ensure that the precedence rules for selecting the winning speculative branch are consistent, even in a schismatic environment. There is no master to pick a winner. Given the same knowledge, all replicas will agree on the same winner. That winner's work will be kept and the work of the other schismatic replicas will be canceled and reported for reconciliation. As work is sent back to the initiating root of the transaction, the speculation is collapsed and a single winning answer is presented.

1.5 Programming an Eventually ACiD Tx

As we shall see, programming for an eventually ACiD transaction is not that bad. There are two kinds of participants:

Resource Participants interact with physical or logical resources of a business that may impact stuff outside of this transaction. Managing shared resources can have its challenges. Using engagements for a business transaction adds a few considerations.

The interaction over engagements uses messaging. Each engagement is a portion of a single business transaction. Sometimes, the resource manager is strongly consistent and not replicated. In this case, the programmer of the resource need not worry about its own resources being schismatic. Replicated resource managers have more complications (discussed below).

Some operations performed by resources are "non-reversible" and require special consideration. Delivering money to an ATM, launching a rocket, and sending a dunning notice should not be taken lightly. We will discuss this more below.

Activity Participants work within the confines of a single business transaction. They cannot access the outside world directly and must work by sending messages over engagements.

Programming activities is easy. They come to life when an engagement is made that invokes them and they have no state. Incoming messages define what they should do. Activities may start outgoing engagements to do more work for the transaction. *Activities must be functionally dependent only on the messages they receive over their engagements.* Nothing else may impact their behavior including time-of-day and/or machine information.

1.6 Dealing with Autonomous Systems

The tree of participants will do work and then wrap up commitments in a Phase-1 similar to a classic 2PC (two phase commit). As a part of Phase-1, each resource manager must also supply a *non-repudiation package*. This will flow back to the other participants (managed by the engagement managers) as a form of guarantee that the resource will do the promised work. In the event the resource manager fails to do what it promised, the non-repudiation package will be presented back to the resource in an attempt to make it right. We will discuss this more below.

1.7 What's Coming in the Rest of the Paper

First, we consider business transactions, how business resources can be shared across them, and the ways in which participants can misbehave. What does it means to access resources shared across transactions? How do the business resources present their business operations to the rest of the business transaction?

Next, we describe the proposed mechanism called engagements, how it connects participants, and some of its special features. We see how the work of a transaction may include different outcomes simultaneously explored in what we call speculative execution. Lest you think the application programmer's job is overwhelming, we explore the constraints imposed on writing a participant in two sections, programming activities and programming resources.

Next, we look at speculative execution as well as how we can resolve the loss of autonomous participants. We look at the impact of engagements across participants and how the business transaction uses engagements. Finally, we conclude and describe our thoughts of where these ideas may be of use in the future.

This paper is simply a thought experiment offered by two guys who've worked in this area long enough to lose their hair. Still, we think that new plumbing can make it easier for applications in this loosely coupled and scary world.

2. THE BUSINESS TRANSACTION

In this section, we examine the business transaction tree of participants, how they are started and finished and how they access shared resources. We examine open-nested transactions, a theoretical basis for access to business resources while not seeing any side effects to the business transaction and show an example. After this, we look at the leaves of the transaction tree, the tree of activities, and consider schismatic and autonomous participants.

2.1 The Business Transaction Tree

In our model, the Eventually ACiD business transaction is a tree.

Roots start and finish the business transaction. Unlike the other participants, the root must be strongly consistent. Roots are neither forgetful (like autonomous participants) nor schismatic. They may be a single strongly consistent system or employ some variant of Paxos [11] to achieve their clarity of mind.

Activities are the non-leaf portion of the tree. They never *directly* interact with any semantic state outside this business transaction. Activities exist solely to coordinate the needs of this piece of work. Anything they need to access from the outside world, they access by interacting with resources, the leaves of the tree.

Resources are the leaves of the tree. Resources are the connection to the rest of the world. They provide the ability to allocate, gain commitments, reserve real-world stuff, and commit or cancel business operations with the rest of the world. See Figure 1.

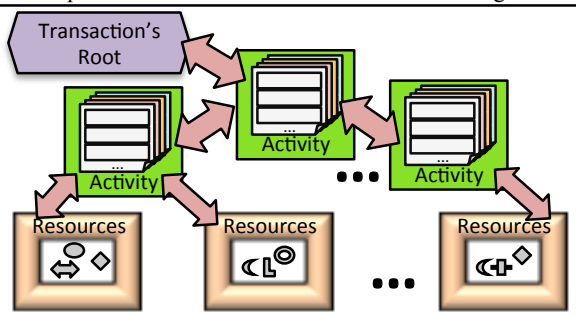

Figure 1) Business transactions start from a root. The root interacts an activity. Activities may interact with other activities and form the non-leaf portion of the tree of participants. Any shared resources are the leaves of the tree.

2.2 Starting and Finishing Business Txs

The business transaction's work arrives from a root. The root uses a single engagement to send and receive messages working with an engagement manager (described below). The engagement manger may run on the root's local system or may run remotely. All of the engagement's work flows through the engagement manager. The business transaction work happens in three phases:

- **Setting expectations (Phase-0):** This is a full-duplex messaging session in which the root describes the desired business outcome and chatters as much as necessary to get the things it wants to get. An example may be scheduling a ten-city trip to Europe. The initial request may not be too selective about the exact timing and order of the visits to the cities. This may evolve as availability of hotels and flights are considered.
- **Getting agreement and picking the outcome (Phase-1):** At some point, the summary of all the proposed work is passed to the root. It may decide to commit or abort this business transaction. If it decides to commit, the proposed changes either commit or abort. This is the commitment of a long-running business transaction whose steps are classic database transactions. If it aborts, this will involve compensating the work of the resources (e.g. canceling reservations at a hotel). Committing may include performing non-reversible actions like launching a rocket or dispensing money at an ATM.
- **Confirming the work is done (Phase-2):** When all the work is actually done, the Phase-2 confirmation is returned to the root.

Between Phase-1 and Phase-2, an autonomous system may renege on its commitment. The engagement manager involved attempts to repair the damage. If this doesn't succeed the root is notified.

Example Business Transaction (from the Root's Perspective):

- I need to schedule a trip to Europe. It is important to be in Paris on August 4th for a business meeting. Side trips to London, Amsterdam, and Berlin of at least two days each are required. I need to be home for my wife's birthday on August 11th.
- Later, a proposed itinerary returns with airline trips, rail trips, hotels, and car rentals.
- The root requests a better hotel in London and proposes to rearrange the dates staying in Amsterdam and Berlin.
- Later, a new proposed itinerary comes back. It looks better and the root says a special command saying: "Let's Do It!"
- As time goes by, the completion of each participant's work for the business transaction is gathered and propagated up the tree. When everything is completed (and the traveler is home), the root's engagement manager has all of the documentation.

2.3 Open Nested Transactions

Open-nested transactions [14] describe the interaction of different layers of abstraction as work is performed. Lower layers of abstraction commit changes to their state as a part of entering into a transaction within higher layer of abstraction. The failure and undo of the higher layer of abstraction does not undo the lower layer transaction but frequently performs a *different* lower layer transaction to return the higher layer abstraction to its desired state (where the higher layer sees the transaction gone). It is accepted that this may yield side effects at the lower layer.

We see this within the implementation of database systems where the logical undo of a transaction's record inserts into a BTree does not undo the block-split caused by the transaction's work. The BTree split is a lower layer and the higher layer transaction may leave side effects when it aborts and that's OK. [17]

2.4 An Example of Not Showing Side-Effects

Doing work with a resource may, in fact, stimulate other work. Consider the following:

Long Running Side Effects Example: the Trip to Europe

1) I schedule a trip to Paris and reserve a hotel room,
2) The hotel notices its occupancy crosses a threshold so the hotel restaurant needs more food,
3) The restaurant increases its order with the grocer,
4) The grocer schedules a delivery with the delivery company,
5) The delivery company realizes its supply of diesel fuel will be getting low so it schedules more fuel, and
6) I change my mind and cancel my trip to Paris.

At this point, the side effects of my reservation *are not* canceled. Each of these business steps has hysteresis. The level to decrease the supplies will be lower than the level that caused the supplies to increase or its business operations would be too jittery.

My short-lived plans for a trip to Europe may cause work and deliveries to happen as a side effect and that's OK.

Resources may have side effects but that doesn't concern the business transaction. The real world has open-nested transactional behavior. Business transactions only interact with the outside world through the leaves of their tree. Side effects are at a lower layer of abstraction. We call the leaves of the tree "resources".

2.5 The Leaves of the Transaction Tree

As mentioned above, any time a business transaction interacts with something visible across business transactions, either logical or physical, the business transaction sees this as a leaf.

Resources and their resource managers do not have side effects visible to the business transaction. They are simply hidden. We assume that resource managers may be either schismatic (due to replication) and/or autonomous.

2.6 The Tree of Activities

Activities comprise the middle of the business transaction tree. The application code for an activity may be defined and named so that an engagement can connect to it. The state of the activity is empty until messages arrive and are processed.

Activities must be functionally dependent only on messages received over their engagements. An activity comes to life when an incoming engagement arrives and delivers messages. The activity may, in turn, make outgoing engagements. These connect it to other activities or to resources.

Compare activities to paper-form-driven work. Historically, an enterprise's work flows through its departments over paper forms and multi-colored replicas. As the work progressed through the departments, more annotations were made and one of the colored papers was torn from the back and stuffed into a file cabinet organized by a sequence number. See Figure 2.

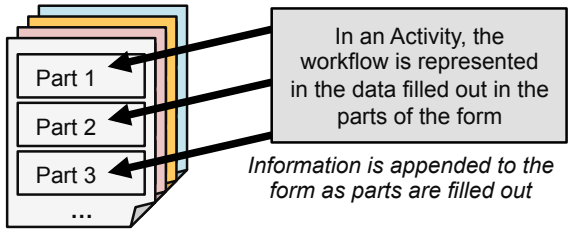

Figure 2) An activity is similar to a classic paper form that traverses departments within (and sometimes across) enterprises. As work is done, it is added to the form. Each participant keeps a record of the work they've seen.

In the classic paper-driven enterprise, each form (with its multicolored replicas) moves around accreting knowledge. Each removed colored sheets has a subset of this knowledge. The form experiences a linear history where additional knowledge is appended to the remaining multicolor master form.

Similarly, activities see one message at a time. They are added to what it's seen. These messages then stimulate outgoing messages.

2.7 Schismatic and Autonomous Participants

Activities and/or resources may be schismatic. This happens when they are implemented as replicas and the replicas process messages in a fashion that results in different results. As we shall see, engagements allow for the resolution of these differences.

Similarly, activities and/or resources may be autonomous. This means they do what they want and may disregard their commitments. Before the two phases to commit the business transaction, an autonomous participant may result in the failure of the business transaction. The real challenges come when reneging happens between Phase-1 and Phase-2.

Engagements and their engagement managers will keep non-repudiation material that proves the promises for work made by the participant. In many cases, the work can be resurrected. In all cases, evidence of the failure can be reported and acted upon.

3. ACCESSING SHARED RESOURCES

This section looks at a set of issues with resource managers and the resources under their control. First, we discuss the business operations that define the interaction with the resources. Next, we discuss how these business operations take the shape of promises and offer isolation to the long-running operations via promises and the predicates describing them.

Following this, we move on to look at the theory behind open-nested transactions and how this lets us provide different layers of abstraction for business operations. These layers of abstraction isolate the business work from its unintended consequences. The rippling effects through logically unrelated things can be ignored and the business transaction can ignore the side effects.

Finally, we examine how non-repudiation offers a mechanism to either clean up the damage caused by an autonomous system reneging from its promises or escalate the damage to the root.

3.1 Resources: Doing Business Operations

As a business transaction executes, it establishes engagements to acquire what it needs to complete its job. The work performed across the engagements describes business operations and these operations deal with the semantic resources of the business.

The message flow over the engagements may include some back-and-forth about the desired results, examining possibilities and making compromises as necessary.

Consider a Trip to San Francisco:

→ I need a hotel near One Market St, SF. I prefer at least 4-star.
← You can have a 3-star across the street or a 4-star a mile away.
→ I'll take the 4-star one and walk a mile.

Note that the interaction *is **not** about read and write of records*! It is rather about the business work being accomplished by this small portion of the business transaction.

3.2 Resources: Leaves of a Transaction Tree

Each resource within the transaction is accessed separately. Imagine a transaction in which you schedule a trip to Europe with two stops in Munich a week apart. If the scheduling of those separate stops was handed to two different activities, there are two different activities talking to the same hotel for two different reservations. The hotel may not see these as correlated and the business transaction sees the separate hotel reservations as different leaves of the same tree. *All access to resources affecting stuff across business transactions is seen as the leaves of the tree!*

3.3 Isolation via Promises

A promise is an agreement between a client application and a service. In [8] and [10] these are called the *promise client* and the *promise maker*. These promises are created using a *promise protocol* that creates an obligation on the part of the promise maker so that the promise client can construct a larger operation.

The promise is an agreement. Typically, the promise maker agrees that the promise client may pick one of a set of possible outcomes. For example, the promise maker may reserve a hotel room for late arrival. As a part of the promise, the promise client has supplied a credit card and agreed to be charged for the first night unless a cancellation occurs with 72 hours notice.

Promise agreements may take different forms and are left abstract. This looseness allows its flexibility. Hotels expose room reservations. Retailers expose products, SKUs, and delivery guarantees. Different promise makers define their own promises.

The same promise maker may use different predicate constraints for different promises. As noted in [8], different resources can fulfill different overlapping promises. For example, room 512 may satisfy both a request for a 5th floor room and another request for a king sized city-view room. The promise maker can (and should) reassign promised resources if that allows more promises. Room 512 may be reassigned to the 5th floor promise and room 620 used to keep the promise for a king sized city view.

We see this in airline seat assignment. The early seat assignments specify an exact seat. Later ones may commit to an aisle seat. Still later, the promise is: "We'll get you on the plane" as the airline happily stuffs you into any seat left open by a no-show traveler. Of course, these promises are not firm as the airline will overbook the travelers and occasionally, you're "out of luck"!

We could express airline seat reservations by their characteristics rather than by specific seat numbers. I'd like a promise that I can have a seat next to my wife, sitting by the aisle, in a seat with lots of legroom. Different seats on the plane may fulfill this promise.

Promises and Isolation: When can the promise maker allow multiple requests with independent predicates to coexist? How can it assign the resources so that all the promises may be kept?

For now, we assume a centralized and reliable promise maker. Below, we will explore schismatic and autonomous ones. The promise maker defines rules for its promise predicates and makes new promises that can coexist with previous ones.

Promises may be refined using the promise protocol. The promise protocol allows promise clients to request promises from their promise maker. These can be renegotiated to increase, decrease, or change the scope of the promise. See Figure 3.

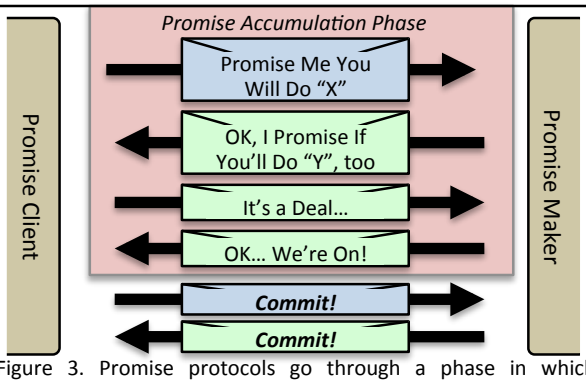

Figure 3. Promise protocols go through a phase in which promises are accumulate (perhaps in both directions). Later, the work is either committed or aborted.

Promises can be bi-directional. While clearly one side of the promise-protocol starts the work, each side can demand promises of the other side to get in a useful result. The protocol completes by committing or aborting the work.

3.4 Open-Nested Transactions

Open-nested transactions extend the classic transaction model in a number of ways. First, there may be nested transactions that are allowed to fail without necessarily failing the parent transaction. Second, when they are open-nested transactions, they may operate at different levels of abstraction within the nested children.

A *classic transaction* is a bunch of operations that appear to execute simultaneously from the perspective of other transactions. The seminal work of Gray and Reuter [7] is a must read.

Nested transactions allow a *parent* to have one or more *children transactions*. These children may abort, having their effects semantically undone while the parent transaction survives.

Open nested transactions are nested transactions in which the children operate at a *completely different level of abstraction*. ACID guarantees are provided at the parent's level of abstraction and show up as multiple independent operations at the child's level. When the parent's level aborts, this may result in the use of different transactions at the child's level to provide the semantic effect of aborting the parent. At the child's level of abstraction, two or more independent operations have occurred. [14][17].

Note that with open nesting of transactions, isolation and consistency *occur at various levels of abstraction*. An attempt to do a transaction at a higher level of abstraction may not appear isolated at the lower levels of abstraction.

3.5 Business Layers of Abstraction

Businesses are complex. Just about every operation performed weaves in and out of many departments and spheres of control. Many times, the business operation driving the change only sees the work from its perspective (i.e. its layer of abstraction).

Levels of Abstraction within My Trip to Europe:

My travel agent deals with airplane tickets and hotel reservations. This is a different level of abstraction than the hotel's concerns about its occupancy and its restaurant's food. My travel agent's work is captured in many discrete "classic" transactions that move the pieces of my trip forward or backwards.

Side effects may exist across the levels of abstraction. Even if we undo work at the higher level of abstraction, the lower level may clearly have consequences remaining.

3.6 You Can't Undo All the Ripples

It is normal in business work for things to fail. It is also normal that the side effects performed at a different abstraction layer to remain after the higher-level operation fails.

3.7 Transactions Don't See Side-Effects

As described above, the business transaction and its operations do not see the many consequences and side effects caused by their work. I may want a hotel reservation and really am unaware of its impact on the restaurant, much less the delivery of diesel fuel to the shipping company for the grocer. This is exactly the same as a classic database transaction. When a transaction aborts, the BTree splits caused by the transaction are not undone. [17].

3.8 Non-Repudiation

Now, we shift to discussing how to cope with resource managers that are autonomous and occasionally renege on their promises.

As a part of the work to commit a business transaction, a resource supplies a collection of data defining its commitments. This is the *non-repudiation data*. As the Phase-1 acknowledgements propagate over engagements up the business transaction tree, the non-repudiation data is propagated, too. When non-repudiation data passes through an engagement manager for an activity and, in turn, up the tree to the parents, it is encrypted by the activity. In this way, by the time the Phase-1 reaches the root, it contains all of the information needed to confirm *all of the promises by all of the resource managers* for the entire tree.

3.9 Autonomous Resource Managers

A resource manager may flake out and decide not to follow through on its promises. The resource manager may also have legitimate reasons for reneging like learning your credit card was stolen since first supplied. To cope, we keep non-repudiation data that offers a proof of the promises made. Re-presenting this may get the work honored. If that doesn't help, we escalate the issue.

4. INTRODUCING ENGAGEMENTS

The business transaction tree of participants (activities and resources) is created starting at the root of the tree. Activities are spawned to manage "per-transaction" portions of the work and resources are contacted to gain access to the rest of the world.

Engagements are the means for communication across the tree and connecting to other activities and resources.

This section discusses engagements, participants, and the work with their replicas. We discuss how causal consistency [1] within a single engagement or arc of the tree offers a framework for dealing with schismatic replicas of a participant. Next, we sketch the challenges faced by the plumbing that we call the engagement manager. Finally, we address the storage within an engagement of non-repudiation data as its participants communicate.

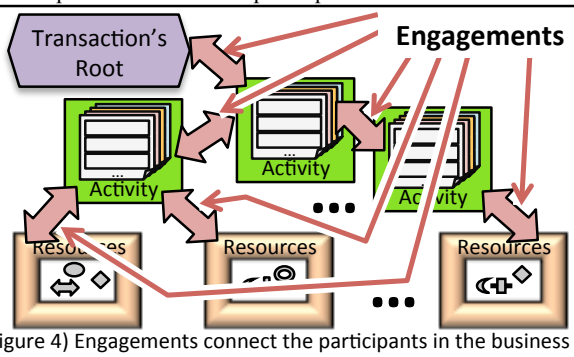

Figure 4) Engagements connect the participants in the business transaction. They capture messages and non-repudiation info relevant to the relationship between the two participants.

4.1 Engagements Connect the Participants

As mentioned above, eventually ACiD business transactions comprise a tree of participants. An engagement connects two of the participants in the tree. If there are N participants in the tree, there will be N-1 engagements. See Figure 4.

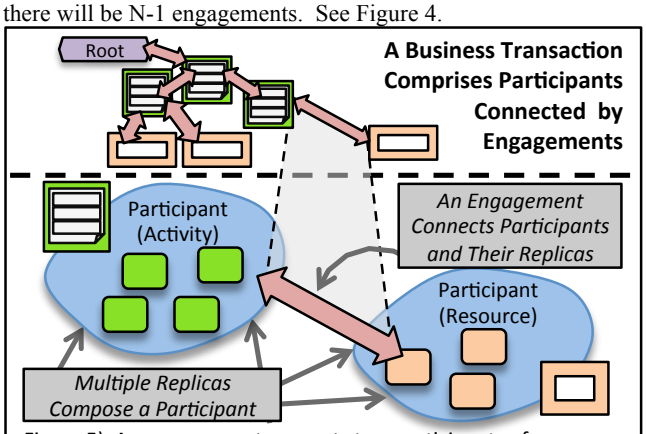

Figure 5) An *engagement* connects two participants of a business transaction. Each of participants may, in turn, comprise a set of replicas running on different machines each sending and receiving messages and state independently.

4.2 Engagements Connect the Replicas, Too

Now, each of these participants may comprise many replicas. The replicas may run at different datacenters or even on different personal devices like your iPhone, iPad, or laptop.

As an engagement connects two participants, it also connects the *replicas of the participants*. Each message is delivered to each replica. When one replica of a participant sends a message, the other replicas see it across the engagement, too. Non-repudiation data is shared across the replicas, too. See Figure 5.

When we say that a replica receives the message, we mean that the engagement manager for the replica receives the message. Below, we will discuss the perspective of the activity or resource application and show how this application code is shielded from the complexity of these messages and their delivery order.

4.3 Sending Messages using Engagements

A message is sent by a single replica and will be received by each replica on the engagement. Every message on the engagement is available to be seen by every replica (either in the same participant or the other participant). Different replicas may send messages at the same time and these may travel past each other.

In addition to messages, a replica may write its non-repudiation data. The messages and non-repudiation data *should* be identical for all the replicas if they are not schismatic.

4.4 Causal Consistency and Schisms

Engagements offer a special form of causal consistency:

If you can see a message at a replica, you can see all the messages visible to the replica that sent the message.

All replicas sharing the engagement (either from the same participant or the other participant sharing that engagement) can see and remember the messages flowing over the shared engagement. Consider the following:

- Message-X is **sent** by Replica-Y at Time-T1.
- Message-X is **read** by Replica-Z at Time-T2.

When Replica-Z reads Message-X, it also sees *every message on the engagement visible to Replica-Y at Time-T1*. The history of this shared engagement is visible to everyone on the engagement.

Not every replica sees everything at the same time. There is not a single view of history. What is guaranteed is that if you can see a message you can see the history *visible to the author of the message at the time the message was written*.

Engagement managers coordinate the version history of the communicating replicas. By tracking when replicas send the same messages in response to the same stimuli, redundant stuttering of twin replicas can be detected and the duplicate messages removed.

While this is a form of causal consistency [1], it is in one way trivial and, in another way interesting. Simply looking at a tree of non- schismatic participants, the communication's causal nature is trivial as work moves down the tree. If you consider schismatic participants, it becomes interesting to look at their messaging as speculative work propagates across engagement managers. Each participant's engagement manager sees the flow of messages for all the possible schismatic replicas.

Each engagement manager at its replica sees all the different permutations of speculative state and the causal flow of messages is seen as a directed acyclic graph. The propagation of actual messages between engagement managers is causally consistent as a DAG of possibilities. Each speculative replica of a participant's application sees its world with a perspective of causal delivery over a tree. That makes it easy for the application developer.

4.5 Replicas and Engagement Managers

Each participant's machine has an engagement manager. The engagement manager sees all the messaging and non-repudiation data for *all the engagements* known to the participant. This includes incoming and outgoing engagements for an activity. The engagement manager has a pretty good idea of the replicas for this participant but that is allowed to be in flux as some replicas may fail and new ones may jump into the fray. See figure 6.

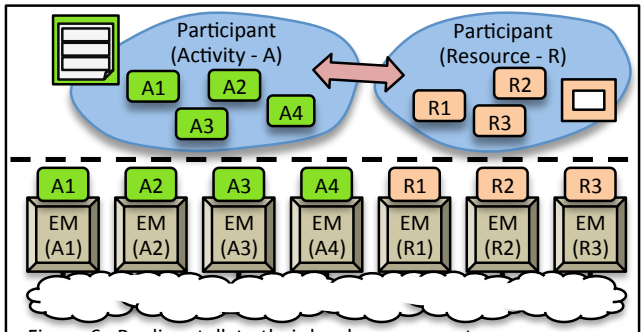

Figure 6. Replicas talk to their local engagement manager. Messaging between the participants and their replicas flows over engagements implemented by engagement managers.

Engagement Managers for Activities: As the first incoming engagement arrives at an activity, the engagement manager is responsible for hooking up a new replica of the activity (which knows nothing other than the messages arriving over that engagement). As messages are received, they are forwarded to the other replicas of this activity. As messages are sent, they are also forwarded to the replicas of the same activity.

When a replica shares its outgoing messages with a replica, it also shares the messages it has received over any of the activity's engagements. So, the engagement manager for a replica handles all of the engagements seen by this replica for the activity. Sometimes, the different replicas have different opinions of the engagements known to the activity. This is eventually shared and all the replicas eventually get the same opinion of the set of engagements, the messages sent and received over them.

As we will see below, sometimes this set of engagements as well as the sent and received messages will represent alternate views of the execution (which we describe below as *speculative execution*). The engagement managers for the replicas see all of these realities as they are shared across the replicas. They are not necessarily made visible to the application code for the activity.

Engagement Manager(s) for the Root: The root of the transaction must be strongly consistent and doesn't have speculative versions. Each root has a single engagement to an activity (which may have schismatic replicas and autonomous proclivities). The root's engagement manager(s) must be strongly consistent either by being centralized or via some form of Paxos[11]. All messages sent to the root over its engagement do not have any perception of the schism that may have occurred in the rest of the participants of the business transaction.

Engagement Managers for Resources: Each resource replica has an engagement manager. Unlike activities, resources have exactly one engagement coming into them. Another difference is that resources may also share business state across engagements, across business transactions, and in a manner that is transparent to the business transaction. Because there is exactly one engagement for each resource within the transaction, there are ways in which we see fewer ordering complexities within the engagement manager for resources. Yet, the interaction with business resources adds its own challenges. We will discuss this more in the section on speculative execution.

4.6 Non-Repudiation Data and Autonomy

Participants may add non-repudiation data to an engagement. Typically, this is encrypted so only systems implementing replicas of the same participant can crack it. This has two uses:

- **Checkpointing to My Replicas:** If I've done some work as a replica, the other replicas may need to know about it. This is a classic checkpoint but with causal ordering. Related messages and non-repudiation data are ordered in their arrival.
- **Non-Repudiation and Resurrection:** Encryption can ensure the non-repudiation data was sent by one of the participant's replicas. This data indicates the stage of the work and functions like a receipt at the dry cleaning store. It cannot be repudiated. This can function as a form of durability if the autonomous system has decided to have "less than durable durability".

Hopefully, by capturing non-repudiation data, the implementation of replicated participants is easier. It provides the means to repair an autonomous participant that doesn't live up to its promises in a business transaction.

4.7 Active versus Passive Replicas

Some engagement managers and their applications may chose to actively process messages at every replica. This means the application runs, consumes the incoming messages, and generates the outgoing messages that are then coalesced when they are identical. *Active replicas* have the advantage of being more responsive and the disadvantages that they consume more computation and occasionally become schismatic.

Passive replicas don't even execute the application but rather have the incoming and outgoing messages recorded and kept ready to replay if necessary to recreate a running replica. They don't go schismatic and don't chew up computation but may take longer to warm up as the messages are replayed.

Active and passive are really somewhat vague concepts. Consider three sibling replicas where one is running at full speed processing, the next is running at 10% the speed, and the third at 1% of the speed. Normally, the third will see its siblings output before it gets around to processing its input. It is (mostly) passive.

The non-repudiation information supplied by an active resource as it does work can mean that the passive replica can catch right up if necessary and continue supporting the same allocation of the same business stuff promised by his big-brother-active replica.

5. SPECULATIVE EXECUTION

In this section, we will discuss the behavior of schismatic participants as they proceed down alternate paths because their replicas do work that diverges in its behavior.

First, we will discuss the ambiguity of whether replicas fire and do schismatic work. Next, we look at some simple cases in which identical work is coalesced. This leads us to looking at the difference between different physical replicas taking different courses of action while separated and the bringing together of the set of messages and state that have diverged.

Next, we look at how we can have a precedence ordering of schismatic work done by resource managers. We look at precedence of work done by activities. How can different replicas in different orders pick the same precedence? With these, we can come up with a single view of the world that resolves the schisms. All that is left is to clean up the effects of the speculative work.

5.1 Firing Up One or More Replicas

This proposal for engagements and engagement managers is agnostic about when and where a participant's replicas fire up and process work. It is fine to have some means to select one active replica and only do the work a single time. It is also fine to have all the replicas fire away and do redundant identical work. Part of

the proposal includes mechanism for the engagement manager to coalesce identical work that has produced identical messages.

As we shall see, the real fun begins when we fire up multiple replicas of the participants and they end up taking different paths.

5.2 Coalescing Identical Work

The engagement manager recognizes that multiple replicas may fire at roughly the same time and perform identical work. Being replicas *of the same participant* is a special relationship. They may do redundant and identical work. This results in the same messages being sent (from the sibling replicas). This is also true for non-repudiation data; identical data is coalesced into one.

When identical messages are sent or identical non-repudiation data is put into the engagement, the engagement manager coalesces these. It verifies that the participating replicas emitted them in the same order. They must have the same history leading up to sending the messages and non-repudiation information.

Identical messages that have been coalesced will only be delivered once to the rest of the engagement.

5.3 Replicating Schismatic Work

Much has been written about replication and eventual consistency [16]. We like looking at this the way *Amazon's Dynamo* [5] does. Dynamo selects availability over consistency as a business choice. This can support schismatic participants.

Dynamo is an *always-writable* store. If any part of the store can accept the write, it will. When a record is read, Dynamo finds the best version it can reasonably locate. This is not necessarily the latest version. In fact, there's no notion of "the latest version".

The arrival of a message into a replica is similar to an always-writable store. If the message arrived, we remember it. If the message is a duplicate of one we've seen then there's nothing new to do. If a message is arriving and we have not yet seen it at this replica, we should do something with it! This scheme is designed to provide availability over consistency even in the face of network partitions. See the CAP Theorem [4], [12].

5.4 Defining Precedence of Resource Answers

Sometimes a schismatic resource manager will give two different answers to the same question. This is natural since the resource managers have to deal with real-world state and the schismatic replicas may have had different real-world state and different experiences before receiving the incoming request for a promise.

To build a self-settling replicated system, it is essential that any engagement manager in the business transaction be able look at two schismatic answers and pick the same one all the time. This means the rules for precedence should be functionally dependent only on the messages themselves, not the specific participant or replica at which the engagement manager is picking a winner.

A proposed set of rules for picking between two answers:

- Did one resource request succeed and the other fail? If so, keep the successful answer.
- Is there an application-defined preference for one answer? E.g. I'll take a room with a queen-sized bed and a view over a room with a king-sized bed but no view.
- Pick some arbitrary aspect of the answers (e.g. their sequence number) and select the lower sequence number.

It is essential to always yield the same result independent of the location or replica that decides on the precedence.

5.5 Defining Precedence of Activities

When an activity is first created, it comes to life because of an incoming engagement. Until it creates outgoing engagements, the only data it receives is from its incoming engagement. So, for a specific activity (or its specific speculative existence), we can order the outgoing engagements by their order of creation.

Activities may diverge from their replicas due to different message delivery order from different engagements. Sometimes, different replicas of an activity will receive two messages each (one per engagement) but the messages arrive in different order. When this occurs, we give precedence to the message received over the first outgoing engagement. Differing speculative execution histories are given precedence in a predictable fashion.

It is essential that this always yield the same result independent of the location or replica deciding on the precedence.

5.6 Precedence, Reordering, and Coalescence

There may be many schismatic and speculative views of this business transaction's work. Different replicas will see different permutations of the choices and independently select the best of what they've seen so far. By working its way up the separate sets of choices, the best of the best should rise to the top.

We need to know that we have consistent rules and that applying part of the work in one replica and another part of the work elsewhere will still resolve to a single choice across the possible speculative universes. *Given the same set of speculations visible to a set of replicas, exactly one choice will dominate independent of the location of processing the subsets as we find a winner.*

Alvaro et al [2] define logically monotonic computation as one that produces the correct answer independent of the order of the inputs. Helland and Campbell [9] define the "New ACID" as "Associative, Commutative, Idempotent, and Distributed". This allows work to be done at least once, anywhere, and in any order. Both papers are describing the same characteristics. These are the characteristics needed to resolve speculation to a single winner even when the resolution is spread widely across many replicas.

5.7 Resolving to a Single World View

During the life of a running business transaction, the engagement managers will notice schismatic work and dynamically resolve it.

Schisms occur for either of two reasons:

- Different answers from a resource promise, or
- Different orderings of message processing in an activity.

When schismatic resource responses are noticed, we calculate the precedence and pick a winner. When schisms are noticed due to message orderings into activities, any engagement manager can prune the loser and all of its descendants. *The precedence can be calculated independently anywhere and yield the same outcome.*

Engagement managers will be constantly pruning speculative paths that are losers compared to some new speculative path. Given any set of possible executions, exactly one will win.

As we package up any message to the root of the business transaction, we will examine a set of alternate histories. Of those, the dominant history is presented to the root for consideration.

It is possible that another speculative history is rattling around some replicas and isn't visible to the engagement manager presenting the message to the root. Of the histories that are visible, the dominant one will be selected as the winner. Anything else will be pruned and its work discarded.

5.8 Cleaning Up Speculative Work

When pruning work, it is easy for an engagement manager to discard a losing version of an activity. The dominant version can continue forward and hopefully commit the business transaction.

On the other hand, cleaning up the losing work for a resource and its promises may be a bit more complex. The resource's engagement manager contacts the resource's application code to cancel or compensate for the promised work. This will hopefully restore the business state to something comparable to what it would have been had the resource never been allocated.

The over-allocation of business resources (e.g. reserving hotel rooms that were not really needed) can clearly impact a business. This is why we believe that the eventually ACiD transaction is not perfectly isolated as it performs long-running work.

All messages seen by the root should resolve to a single view of the world. By Phase-1, we have also selected exactly the set of resources we hope to commit to the business transaction.

6. PROGRAMMING ACTIVITIES

This section describes the issues faced by an application programmer writing the application code for one of the activities within the long-running business transaction.

We first talk about constraining the behavior to being dependent on incoming messages and their order. Next, we consider the incoming engagement that starts the activity within the business transaction. This takes us to the creation of outgoing activities and how each replica labels its outgoing activities. Next, we talk about how schismatic activities are unaware of their evil twins. Finally, we consider one piece of advice to reduce (but certainly not eliminate) some of the causes behind schismatic activities.

6.1 Activities as Functional Computation

Each activity and its programmatic behavior must be functionally dependent only on the messages that come into it. The application programmer for the activity has no worries about concurrency, speculation, or any other crazy behavior. It simply must follow the constraint that it never looks at anything other than the messages coming over the engagements for this activity.

6.2 Activities: the Incoming Engagement

An activity is always started by exactly one incoming engagement. Since the activity's behavior and outgoing messages are *functionally dependent on only the incoming messages*, we see the replicas of the activity starting out running as identical replicas consuming and emitting the same messages.

6.3 Activities: Outgoing Engagements

When an activity creates an outgoing engagement, it gives the engagement an *Engagement-ID*. This Engagement-ID must be functionally dependent only on incoming messages. You can't consider time-of-day or ask the operating system for a unique-id.

As long as the replicas are working on the same messages in the same order, the outgoing engagements created by this activity will have the same Engagement-ID. When two replicas make engagements with the same Engagement-ID, the engagement manager can coalesce them into the same engagement.

By coalescing the outgoing engagements from different replicas into the same engagement, identical messages sent over those outgoing engagements can be coalesced and the speculative alternate universes brought back into a single outcome.

6.4 Living in Your Own Happy World

Different replicas can diverge in their behavior. There are two reasons this can occur. First, messages from different engagements may be processed in different orders due to racing of the message delivery to the replicas. Second, different requests to resources may get different answers. Usually, two requests to get a hotel room will result in the same answer (e.g. two separate reservations for a king-sized non-smoking room) but sometimes, you just get unlucky and get different answers. Schisms only occur when the participant is actively processing the messages (and generating output). Passive participants just wait for both input and output messages and, hence, don't cause trouble.

Eventually, all the engagement managers for the different replicas will notice the divergent message orderings. There may be divergent incoming messages (because the other participant went schismatic) or divergent outgoing messages (because the ordering of the incoming messages processed was different). Either way, different paths though the universe may accumulate at an engagement manager. The engagement manager will show the application exactly one history and present messages for that. The programmer for the activity's application won't give a darn about the divergence but will, instead, blithely continue on unaware that there alternate universes either actively or passively present.

6.5 Reducing Schism

Activities may send messages on many different engagements without waiting for replies. The answers to outgoing messages may race and come back at different times. Different replicas of an activity may receive message responses in different orders.

Without care, this could cause different behavior from the different replicas. The different order can cause different work to be initiated and the replicas can diverge unfettered.

When a replica issues parallel requests on different engagements, the application programmer *should consider* waiting until all the responses have been received before listening to any response. This can *reduce* the permutations by which the replicas diverge. Of course, this is not always practical.

7. PROGRAMMING RESOURCES

This section describes the challenges faced by the application programmer that develops a resource manager capable of providing resources to a business transaction.

First, we look at how resources perceive their incoming engagements. Next, we consider some of the challenges seen by replicated resources, especially when they take independent and potentially schismatic action. We look at stuttering engagements with different forms of schisms and how they affect the allocation of resources and these are resolved. The next portion looks at probabilistically allocating resources, over-booking versus over-provisioning, and the cost of selective consistency. All of these real-world problems inject themselves into our world of long-running business transactions. Next, we present a discussion of how to predictably coalesce resource schisms. Finally, we conclude the section on programming resources by considering Phase-1 notifications and then Phase-2 notifications and how they interact with non-repudiation and autonomous resources.

7.1 Resources and Incoming Engagements

When a business transaction (or more precisely an activity within a business transaction) needs to interact with the business itself, the activity creates an engagement to a resource that controls what is needed. When creating the engagement, the activity may specify stuff that narrows down the desired resource.

I'd like to book a room at the Hyatt Regency San Francisco.

The engagement created by the activity may very well connect to the resource manager for the Hyatt Hotels but adds information routing it to the California division of the reservation system.

When the incoming engagement fires up the resource, it is aware of the stuff narrowing the needs (e.g. a California Hyatt) as well as incoming messages from the activity that started the resource.

Unlike activities, the resource participants are tapped into state that is shared within the business as it works to accomplish this single business transaction. Each incoming engagement will set out to negotiate the allocation of some resource, work, or commitment that impacts the business. If a single business transaction issues two engagements to access resources, they appear as two separate operations to the resource manager and, from the standpoint of the business transaction, are independent.

7.2 Replicated Resources

Sometimes, the implementation of a resource participant is, itself, replicated and potentially schismatic. This presents different challenges than a replicated activity.

Replicated resources are somewhat easier in that they are only concerned about a single engagement for the business transaction. That means that they don't need to worry about messages coming in an out of different engagements. On the other hand, resources have additional problems that are perhaps more challenging.

Unfortunately, the allocation of stuff shared by other business transactions requires interaction with that shared stuff. If the participants implementing resources were not replicated, they could have something like a shared database against which they could look to see what is available to commit and make an allocation for this engagement. That would be relatively simple.

In this more complicated world, the incoming engagements have messages that may hit the separate replicas of the resource managers independently. Either they coordinate across the replicas (essentially providing strong consistency) or they act independently and attempt to make promises based solely on the business stuff known to the separate replicas.

7.3 Stuttering Engagements and Resources

Engagements between activities and resources may be complex.

Consider the following possibilities:

- **The activity starting the engagement has a simple schism.** Two engagements are started with identical Engagement-IDs. The sending activities are walking through the same path in the universe and will send identical messages. They will remain identical unless they receive divergent messages into them. The engagements (and the messages across them) will be coalesced by every engagement manager that sees the matching Engagement-IDs. This will happen at each replica of the resource (when it sees the stutter) and also at the replicas of the activities (when they see the stutter).

  When this occurs, the resource manager (and its engagement managers) will attempt to process this as one request for one business operation. Of course, as discussed below, the resource manager itself may be schismatic.
- **The starting activities have a complex schism.** This happens when flow of schismatic work has happened in a way where truly different behavior (rather than repeats of the same behavior) has happened. In this case, the resource manager will treat the business operations as completely separate requests and attempt to fulfill them all.
- **An arriving engagement hits multiple resource replicas.** When this happens, we have a number of possibilities. The resource replicas could be active and passive. This means that usually, we allocate just the needed business resource for the business operation arriving on the engagement.

  Sometimes, we have more than one active resource manager replica. As the incoming engagement and its promise requests are processed, we may over-allocate business resources in our zeal to avoid coordination across replicas. Sometimes one replica succeeds in allocating resources and another fails. As the replicas exchange knowledge through their engagement managers, any difference should be reconciled by calling into application code for the resource application. This is an additional complexity for resource manager applications.

  Note that this is inherent in loosely coupled resource managers. Because they both manage shared stuff (e.g. inventory) and may fire independently to process the same request for promises, they will need to reconcile conflicts in their answers.

7.4 Probabilistically Allocating Resources

Consider a participant managing an enterprise's physical (or logical) resources. Since this participant may be schismatic, we allow for the resources to be allocated with partial knowledge of what its replicas have done. This takes a risk that allows faster responsiveness and higher availability. Alternatively, we must be overly conservative in parceling out the resources.

7.5 Over-Booking vs. Over-Provisioning

If an independent replica of the resource manager is allowed to promise business-resources, it must have a set of rules and a bunch of allocated business-resources to promise. The resource manager (i.e. the replicas managing the business-resources) may:

- **Over-Book:** This happens when a business-resource is promised without certainty that it is available. "Last time I heard from my replicas, there were plenty of widgets… This should be OK". If there are 3 replicas and 6000 widgets, each may allocate up to 6000 if it really wants to do so.
- **Over-Provision:** There're 3 replicas and 6000 widgets. Each may allocate up to 2000 without coordinating. This will ensure conservatism in the promises. It may also mean we fail to accomplish work by being too safe.

This assumes independent replicas. While we are exploring this independence, there are also options for selective consistency.

7.6 Selective Consistency… What's the Cost?

Resource managers may have a sliding scale of consistency. Sometimes, a replica takes independent actions. Other times, tighter coordination is needed.

When you deposit your brother-in-law's check for $100 into your branch, there is no hold placed on the check. When you deposit his check for $20,000, either there will be a hold or your branch with coordinate over the phone to ensure the funds are available.

The consistency rules depend on the size of the check!

A replica may have business-resources that it can allocate and promise unilaterally. If it has insufficient business-resources to fulfill the promise, it can coordinate with its replicas to see if the resources are available anywhere in the combined set of replicas.

Don't deny a promise request without trying to coordinate:

If a single resource-manager replica cannot find what it needs in its own pool, it may check with other replicas. This is an OK business strategy but may mean the promise-request takes a long time to satisfy. This is a consistency versus availability choice.

7.7 Predictable Resource Coalescence

Sometimes, multiple replicas will allocate resources for the same promise request. When this happens, the working state for the two promise-responses will be different. All of the schismatic replicas receive information about their siblings' working state.

As we coalesce two promises, *the same winner and loser* must be chosen. This is essential to ensure the selection of the winner is idempotent. If two replicas look at the redundant resource and each cancels a different resource, we'd get a mess.

This provides eventual consistency. Eventually, it's the same!

The algorithm for picking the winner or loser may depend on the actual resources promised. For example, "I'll keep the aisle seat and cancel the window seat". In the absence of resource specific criterion, the winner must depend on some other artifact of the working state that offers the *same result wherever it is calculated*.

7.8 Non-Repudiation and Phase-1

Whenever a resource promises to do something, we expect that the promise will be accompanied by a special collection of data known as *non-repudiation data*. Engagements have special provisions for this data to be supplied and it is not a message in the classic sense. Typically, the non-repudiation data is encrypted in a fashion that makes it easily cracked by other replicas of the same resource manager even later in time.

When a promise is made, the non-repudiation data should be supplied. By the time we consider a Phase-1 for the long-running business transaction, all promised resources should have this.

If a resource manager decides to renege on its promise, the non-repudiation data is wrapped up in multiple layers of encryption but is available to the root. Each layer of the business transaction tree has wrapped up and encrypted all the promises made below it.

If needed, the business transaction can reincarnate any part of the business transaction activities need to re-drive the work. Remember, an activity is just code and a sequence of messages fed to it. Interim state for the activity can be captured when it choses to (or not to) supply non-repudiation data as an activity. By encrypting at each level, we don't violate any encapsulation needed in a loosely coupled distrusting collection of machines. Eventually, all the state of the business transaction is smashed together and held at the root's engagement manager.

7.9 Non-Repudiation and Phase-2

The business transaction (as knit together with engagements) may live a long time. If I schedule a trip to Europe and book the trip in March but take the trip in September, Phase-2 of the business transaction will gradually occur in September as I complete my various hotel stays and airline trips. In the meantime, the engagements, activities, and resources allocated live in a collection of records in various databases spread across independent and autonomous systems.

Until I actually stay in the hotel in Paris, my non-repudiation data is pretty important to me. While there may have been intermediate booking agencies that prepared the reservations, I want and need the ability to cry foul if there's no record of my stay when I arrive at the hotel lobby. The long-lived nature of this business transaction, the ability to nest promises under activities, and eventually complete the work (or present non-repudiation data) are very important. Only after the work itself has happened (and Phase-2 messages are sent over the engagements) does the business transaction complete.

8. SOME MISCELLANEOUS THOUGHTS

Engagements offer a somewhat different way of thinking about long-running work and how it can be delivered. We are arguing that an approach that this allows for *better service and SLAs* with *crappier servers and components*.

In this section, we first look at how the relationship between active and passive replicas can provide a soft real-time behavior providing a higher chance of meeting a deadline. Next, we consider the consequences of non-repudiation and message storage as it allows a participant to be very forgetful by depending on the rest of the business transaction tree.

Following this, we look at the implications of high-failure rates across schismatic participants. Because we can cope with failures and schisms, we can function well with unreliable servers. It's better to have more cheap servers than a few reliable ones!

Finally, we consider what it means to integrate legacy servers with all their strengths and weaknesses into a business transaction.

8.1 Timeliness, Timers, and Redundancy

Engagement managers deliver engagements and messages to replicas of a participant. This can be used to manage SLAs (Service Level Agreements) in a soft real time fashion.

Normally, the active replica does its work within an acceptable window of time and its passive siblings replicas see the *results of the work* at the same time as they see the *request for the work*. When this works, we avoid any speculative execution with its costs of redundant work and over allocation of resources.

When the active replica takes too long, an engagement manager can speculate and find another replica to get the work done in the agreed upon time frame. In this fashion, the probability of meeting the deadline can be significantly improved while engagements provide the framework for cleaning up the mess.

8.2 Taking the Low-Road on Durability

There's nothing wrong with an activity participant implementing its state in a fashion that may get destroyed in a system failure. While it may not be the best performing choice (as contrasted with keeping its state in a local database), it should be correct.

This is not as obvious an option for a resource participant since they will typically have their own shared business state that will not be captured in the non-repudiation data.

8.3 Taking the Low-Road on Clusters

Similarly, the systems implementing activities may take a casual attitude towards the individual servers' availability. By having a bunch of servers (each with mediocre availability), we can have high aggregate availability. When used for activities, this will be just fine since the engagement mechanism copes with a replica failing as well as schismatic behavior.

When implementing resource managers over a cluster of flakey servers, that's OK, too. You can choose to have a Dynamo-esque [5] replication that ensures a sufficient number of replicas exist for a resource manager at the risk of slight ambiguity in the correctness of the results of the promise operations. This is just like the classic over-booking ambiguities for many systems.

8.4 Dealing with Legacy Servers

Suppose we have a legacy application server on a mainframe or some other server. How can it work with eventual ACiDity?

It is viable to implement an engagement manager that runs on the legacy system. It may also run on another server sitting next to the legacy system. The legacy server takes on the role of a

participant in the loosely coupled transaction. The specific role (i.e. activity or resource) depends on the application running on the server. Messages into and out of the legacy server are captured and the sent to the application. It is presumed that messages may be retried and, hence, the application is idempotent. To nest the legacy system, we may need a compensating transaction, if indeed the legacy system does not allow itself to be subservient.

9. CONCLUSION

This paper is a thought experiment. We believe that fundamental changes have happened in distributed systems:

- Business transactions are getting more complex: They span more disparate and autonomous computers than ever before. Previously, complex business work moved from computer to human to computer, etc. The intervening humans resolved a lot of irregular behavior.
- Autonomous systems are normal: Nobody wants to trust someone else's computer tying up their resources. Surely, not their locked records! Business resources (e.g. inventory) may be held for a while but only to make a business deal work.
- Schismatic replication is a fact of life: People do work across occasionally connected devices (e.g. phones). Teams have many people advancing workflows in schismatic directions. Cloud-based computation now runs across many datacenters. Businesses may choose availability over consistency [4].

These fundamental changes call for new forms of work.

We propose a perspective on business work called *eventually ACiD business transactions*. This is a type of long running work with open nested transactions, the use of the promise protocol with promise-based isolation, and a tree structure. Shared resources are constrained to the leaves of the tree.

This paper also proposes a new communication mechanism called *engagements*. With an engagement, any member may write anything it wants. The other members see it eventually.

Engagements offer the following guarantees:

- Subjective history: As you see a partner's message, you see the partner's view of the engagement when it wrote the message.
- Justification to replicas: Encrypted evidence may be sent to replicas to privately explain what happened.
- Non-repudiation: Evidence written by a participant may be tossed in its face if an autonomous system bails out.
- I'm flakey… Remember for me! Encrypted evidence may be a form of durability. A participant may pass stuff it needs for "durability" to other engagement members.

Using these proposed mechanisms, we have suggested some possible improvements to the bleak picture we've painted:

- **Eventual Atomicity:** We may have missing parts of the business transaction due to the flakiness of autonomous participants. We may have extra work done due to schisms. Eventually, the missing pieces are replaced (or the system escalates the problem). Eventually, extra work is coalesced.
- **Eventual Consistency:** Once all the work is done, the business rules are met and all the replicas agree (or an escalation occurs).
- **Probabilistic Isolation:** Since autonomous behavior can cause repairs late in the transaction, we cannot guarantee two-phase-locking [3][7]. Hence, we cannot guarantee isolation. However, the use of open nesting to semantically isolate side effects and the promises protocol to allow predicate-based locking will dramatically reduce conflicts.
- **Eventual Durability:** While participants may be autonomous and just toss durability out the window, a tree shaped transaction based on engagements can catch evidence along the way (at the partner systems) and be used to re-drive the work. Either we will get the parts of the transaction to durably stick or we will complain and escalate (potentially to humans).

Hopefully, we can use some of these techniques to broaden our coordination of useful business work while lowering the demands and expectations on the participants.

10. ACKNOWLEDGMENTS

We thank Jeremy Horwitz, Shel Finkelstein, Joe Hellerstein, Ian Varley, and our anonymous reviewers for their great input.

11. REFERENCES

[1] Ahamad, M., Neiger, G., Burns, J.E., Kohli, P., Hutto, P.W. (1995), Causal Memory: Definitions, Implementation, and Programming. *Distributed Computing*, 9(1).

[2] Alvaro, P., Conway, N., Hellerstein, J., Marczak, W., (2011) Consistency Analysis in Bloom: a CALM and Collected Approach. *CIDR 2011.*

[3] Bernstein, P.A., Hadzilacos, V., Goodman, N. (1987) Concurrency Control and Recovery in Database Systems. *http://research.microsoft.com/en-us/people/philbe/ccontrol.aspx*

[4] Brewer, E. A. (2000). Towards Robust Distributed Systems (abstract). *ACM Symp on Principles of Distributed Comp.*

[5] DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., Vogels, W. (2007). Dynamo: Amazon's Highly Available Key-Value Store. *Proc. of the 21st ACM SOSP*

[6] Garcia-Molina, H., Salem, K., (1987). Sagas. SIGMOD.

[7] Gray, J., Reuter, A. 1992. Transaction Processing: Concepts and Techniques. *Morgan Kaufman*

[8] Greenfield, P. Fekete, A. Jang, J., Kuo, D. Nepal, S. (2007) Isolation Support for Service-Based Applications: a Position Paper. CIDR 2007.

[9] Helland, P., Campbell, D. (2009) Building on Quicksand.. *CIDR 2009.*

[10] Jang, J., Fekete, A., and Greenfield, P. (2006). Delivering Promises for Web Service Applications. *Univ of Sydney School of Information Technology. TR-605.* Dec 2006.

[11] Lamport, Leslie (2001). Paxos Made Simple *ACM SIGACT News (Distributed Computing Column)*

[12] Lynch, N., Gilbert, S. (2002). Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services. *ACM SIGACT News, Vol. 33, Issue 2, pg. 51-59.*

[13] Mohan, C., Haderle, D., Lindsay, B., Pirahesh, H., Schwarz, P. ARIES: a Transaction Recovery Method Supporting Fine-Grained Locking and Partial Rollbacks using Write-Ahead

[14] Moss, J.E.B (2006), Open Nested Transactions: Semantics and Support. *Workshop on Memory Performance Issues. http://www.cs.utexas.edu/users/speedway/DaCapo/papers/wmpi-posters-1-Moss.pdf*

[15] Reuter, A., Wachter, H. (1991) The ConTract Model. IEEE Data Eng. Bull. 14(1)

[16] Vogels, W. Eventually Consistent. 2008 *ACM Queue.*

[17] Weikum, G.; Schek, H. 1992. Concepts and Applications of Multilevel Transactions and Open Nested Transactions. *Database Transaction Models for Advanced Applications*

[18] Wikipedia. Speculative Execution.