Institute for Software Research

University of California, Irvine

Extending the REpresentational State Transfer (REST) Architectural Style for Decentralized Systems

Rohit Khare
Univ. of California, Irvine
rohit@ics.uci.edu

Richard N. Taylor
Univ. of California, Irvine
taylor@uci.edu

October 2003

ISR Technical Report # UCI-ISR-03-8

Institute for Software Research
ICS2 210
University of California, Irvine
Irvine, CA 92697-3425
www.isr.uci.edu

Extending the REpresentational State Transfer (REST) Architectural Style for Decentralized Systems

Rohit Khare & Richard N. Taylor
Institute for Software Research
University of California, Irvine
Irvine, CA 92697-3425
{rohit,taylor}@ics.uci.edu

ISR Technical Report # UCI-ISR-03-8

October 2003

Abstract:

Because it takes time and trust to establish agreement, traditional consensus-based architectural styles cannot safely accommodate resources that change faster than it takes to transmit notification of that change, nor resources that must be shared across independent agencies.

The alternative is *decentralization*: permitting independent agencies to make their own decisions. Our definition contrasts with that of *distribution*, in which several agents share control of a single decision. Ultimately, the physical limits of network latency and the social limits of independent agency call for solutions that can accommodate multiple values for the same variable.

Our approach to this challenge is architectural: proposing constraints on the configuration of components and connectors to induce particular desired properties of the whole application. Specifically, we present, implement, and evaluate variations of the World Wide Web's Representational State Transfer (REST) architectural style that support distributed and decentralized systems.

Extending the REpresentational State Transfer (REST) Architectural Style for Decentralized Systems

Rohit Khare and Richard N. Taylor
Institute for Software Research, University of California, Irvine
{rohit,taylor}@ics.uci.edu

ISR Technical Report # UCI-ISR-03-08

Abstract

Because it takes time and trust to establish agreement, traditional consensus-based architectural styles cannot safely accommodate resources that change faster than it takes to transmit notification of that change, nor resources that must be shared across independent agencies.

The alternative is decentralization: *permitting independent agencies to make their own decisions. Our definition contrasts with that of* distribution*, in which several agents share control of a single decision. Ultimately, the physical limits of network latency and the social limits of independent agency call for solutions that can accommodate multiple values for the same variable.*

Our approach to this challenge is architectural: proposing constraints on the configuration of components and connectors to induce particular desired properties of the whole application. Specifically, we present, implement, and evaluate variations of the World Wide Web's REpresentational State Transfer (REST) architectural style that support distributed and decentralized systems.

1. Introduction

We are interested in designing decentralized software for a decentralized society — systems that will permit independent citizens, communities, and corporations to maintain their own models of the world. Portions of such applications must operate under the control of multiple, independent administrative authorities (agencies); and may be physically separated to the extent that communication latency between those parts becomes a significant factor in their design.

The state of the art in software engineering has long focused on designing solutions for *distributed* systems, where multiple agents share control of a single model. Unfortunately, this presumes it is always possible to establish consensus over the current value of a variable. Physics abandoned simultaneity with relativity in 1905; formal models of computing disproved consensus over faulty, asynchronous networks in 1985 [14]. Regardless of how dominant centralized client/server architectures may appear to be today, the physical limits of latency and the social limits of free agency will make decentralization inevitable for software as well.

1.1Scenario

Electric utilities are an example of an industry that is embracing decentralization, and could well motivate development of novel architectural styles.

The origin of the hardware/software system for controlling electricity distribution is essentially centralized [8]. Local, regional, and international grids are interconnected through single-agency control centers, typically with a public mandate to preserve equilibrium supplies at all costs (i.e. they are independent of any private generator or consumer).

By contrast, one widely acknowledged vision of the future of power generation, distribution, and storage is to support a vast array of much smaller-scale devices, potentially owned and operated directly by consumers and businesses [5] in the absence of central oversight. Device control software might support such independence by replacing references to a "price" variable with estimates of the best price available from its neighbors, creating a network of small-scale markets.

However, eliminating a single point of failure also eliminates a single point of control. Today, hierarchically delegated regulatory authority establishes an agency responsible for maintaining reliable power. Control software for decentralized 'power webs' not only needs to establish interoperability standards [9], but also needs to manage dynamic, peer-to-peer trust relationships.

1.2Approach

Our approach to coping with uncertainty and disagreement is based on software architecture: constraints on configurations of components and connectors that induce desired properties of an overall system. This paper introduces several new architectural styles that are expressly designed to accommodate decentralization. First, we sketch a formal model of the problem that

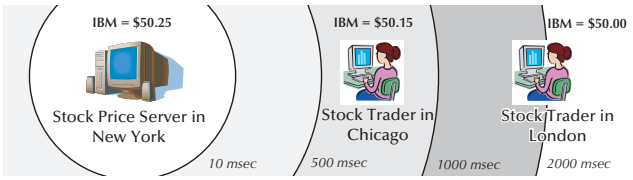

Figure 1: Latency induces uncertainty for traders "further" away from a centralized resource.

allows us to analyze the limitations of consensus-based architectural styles. Second, we address those limitations by identifying new architectural elements and constraints that could be added to an existing network-based architectural style to induce each of those properties. Third, we evaluated the feasibility of those newly derived styles by implementing infrastructure for, and applications of, each.

2. Problem Analysis

Like any other design discipline, software development is subject to the vagaries of fads and fashion. In recent years, there has been a surge in the popularity of the term 'decentralization': we hear of 'decentralized file-sharing,' 'decentralized supercomputers,' 'decentralized namespaces,' and a slew of 'peer-to-peer,' 'Internet-scale,' and 'service-oriented' architectures [10, 32, 33].

To date, the software engineering and software architecture literature has not embraced a formal definition of 'decentralization.' Indeed, in a full-text search of the ACM Digital Library, we found that it was often considered a synonym for 'distribution' until only recently. Even as of 1998, it only occurs in the official ACM subject classification once, and then only with respect to the organization of MIS organizations [1].

Thus, our first goal is to provide precise, testable definitions. In this section, we will discuss the factors leading to decentralization (latency and agency); provide a formal definition of simultaneity in terms of the consensus problem; and use that to define the properties of centralized, distributed, estimated, and decentralized resources.

2.1Latency

Latency makes simultaneous agreement impossible in many real-world situations. Consider how it affects a stock traded on the (centralized) NYSE market. A stockbroker in London interested in knowing the current price of the stock consults a server that broadcasts its current price. Realistically, Internet delays could range up to two seconds, or worse. If the actual price in New York were changing at up to 1 Hz, it would become impossible for the London stockbroker to know its *current* price.

The concentric circles in Figure 1 represent latencies (on a logarithmic scale). Their radii are correlated with distance, but more intriguingly, also determine the maximum update rate of an event source. We call this the 'now horizon', since it demarcates which components can reliably refer to the value of a variable 'right now.'

Latency is an absolute constraint for software architects because it takes time and energy to transmit information across a channel. It can be factored into three separate limits: propagation delay, bandwidth, and disconnection (longest tolerable interruption).

2.2Agency

While latency is a physical limit, the concept of an agency is a socially constructed one. We are referring to the divergent interests of the organizations that ultimately own and operate the computers that software runs on [37].

An agency boundary denotes the set of components operating on behalf of a common (human) authority, with the power to establish agreement within that set and the right to disagree beyond it.

Consider a database package. Run a 'local' copy for yourself, and then if you store `x=5`, then `5` is the one and only true value of `x`. Accessing the same application over a network, though, raises the possibility that the data may have been tampered with or biased. That is a profound difference between the output of the local database component ("*I* believe `x` *is* `5` now") and that of a remote database *service* ("Someone *else* claimed `x` *was* `5` then").

2.3Simultaneous Agreement

> The impossibility of consensus is considered to be one of the most fundamental results of the theory of distributed computing [30].

In 1985, Fisher, Lynch, and Paterson proved that on a completely asynchronous network (one with maximum message latency $d=\infty$), if even one process can fail, then it is impossible for the remaining processes to come to agreement [14]. Lynch's textbook also includes a proof that even with a partially synchronous network (one with only a finite *d),* but with message loss or reordering, consensus still requires at least d seconds [30]. In general, tolerating f processes failing requires at least $(f + 1)$ additional rounds.

Event so, this model of consensus still does not ensure simultaneity. Some processes may decide sooner than others, because of varying network latency. Furthermore, if the value being shared is modified, some processes may keep using the prior value after others have moved on.

The term "simultaneous agreement" originated in contract law, specifically for defining financial instruments. An ordinary cash transaction is a simple example of simultaneous agreement upon a trading price, P_{trade}.

Restating this formally as a condition on two separate variables, a leader and a follower, we define simultaneous agreement as the interval satisfying this conjunction:

$$\exists\, t_0, t_i, t_j : (t_i \le t_j) \wedge (t_i \le t_0 + d) :$$
$$\forall\, v : t_i \le v \le t_j : \; P_{leader}(v) = P_{follower}(v)$$
$$\wedge\; \forall\, u : t_0 \le u \le t_j : \; P_{leader}(u) = P_{leader}(t_0)$$

That is, the leader and follower values must become equal at some point; and the leader must not have changed in the meantime. This condition is met in the shaded area of Figure 2, where world-lines are drawn vertically for the state of the leader and follower processes.

Since a message takes at most d to travel, simultaneous agreement only holds during the interval after the message arrives until P_{leader} changes from 5 to 3. However, if the variable changes after an interval shorter than d, as from 3 to 7, it is impossible for the message to arrive "in time"; and hence simultaneous agreement is also impossible. A corollary is that it is impossible to guarantee simultaneous agreement for any centralized resource that changes more frequently than $1/d$ times per second (or $1/(f+1)d$ times per second, in order to tolerate f failures).

Finally, note that our definition requires P_{leader} to remain constant while agreement is established. At the bottom of the diagram, even though the leader switches back from 7 to 3 before the message "3" arrives at the follower, that is mere coincidence. Either a follower must contact the leader and request a lock, or the leader must indicate the period of time it commits to holding a value constant, which is called a lease.

A leased value is represented by a *(value, interval)* pair, where *interval* is specified by the start time and duration of the lease. In conjunction with a global clock, the lease makes it possible for the recipient to determine whether the value is still valid. In Figure 2, the follower can correctly reset the value of $P_{follower}$ to $\varnothing$ at the first instant the leader is able to change (in this case, from 5 to 3) if we extend the message to specify its lease interval.

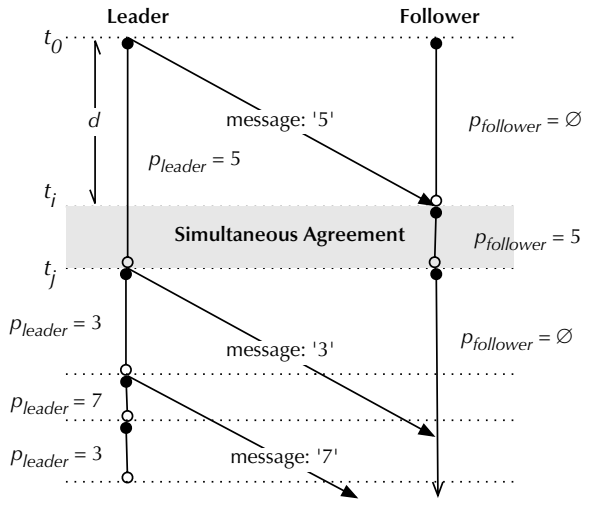

Figure 2: The shaded region illustrates an interval of simultaneous agreement.

2.3.1 Related models. Since consensus is such a fundamental problem, it is not surprising that software architects also have a variety of other models to choose from:

Byzantine Generals. In this problem, several parties must coordinate an attack simultaneously to win — but can also lie to each other [29]. The solution originally proposed is only robust against conspiracies of up to $1/3$ of the generals. This can still suffice for read-only decentralized applications, such as file sharing [26].

Invariant Boundaries. First identified in [6] as the boundary between systems that agree on an invariant and those that cannot, it closely resembles our specific condition of simultaneous agreement. A more general condition applying to invariant boundaries was stated as the Consistency, Availability, Partitionability (CAP) theorem in [15] later proven in [17]), which states:

> It is impossible to reliably provide atomic, consistent data when there are partitions in the network. It is feasible, however, to achieve any two of the three properties… most real-world systems today are forced to settle with returning "most of the data, most of the time."

Logical Clocks. Over the years, there have been many approaches that simulate the operation of a centralized, sequential processor atop a distributed processing network: logical clocks [27], virtual synchrony [4], and group communication [34].

Single-Assignment Variables. An architect could avoid disagreement entirely by restating the problem so as to eliminate mutable variables [38]. One example is the technique of single-assignment: rather than resetting the value of P_{leader} several times, a series of distinct variables could each be set just once: *First-*P_{leader}, *Second-*P_{leader}, and so on. Another approach is manipulating pointers to "future" values in parallel functional languages [19].

Peer-to-Peer Communication. [16] presents a formal model for peer-to-peer computing that uses variables to represent channels between peers. Casting communication channels as variables may make it clearer that the latency of network links also determines the maximum possible update frequency of any interaction across them.

2.4Definitions

Using our formal model of simultaneous agreement, we derived precise, declarative definitions of the properties of centralized, distributed, estimated, and decentralized variables in an expanded work [23]. For this paper, we opt for a more concise, if less formal, definitions:

> A **centralized** variable requires simultaneous agreement between a leader and all of its followers.
>
> A **distributed** variable is determined by applying a shared decision function over all participants' inputs.
>
> An **estimated** variable is in simultaneous agreement only a fraction of the time.

A **decentralized** variable is determined by applying a private assessment function over other trusted participants' variables (or estimates of those variables)

Another way of distinguishing these terms is by the degree of indirection required to implement each. The basic element of information storage is the *value*: centralization requires every agent to use the same value. This can be accomplished as simply as by connecting several devices to the same wire or other shared medium.[1]

The next level of indirection is a private namespace for storing values over time: the *variable*. In a distributed system, a closed group of agents uses a single logical name to refer to a shared variable — even though its actual value is not stored in one place, but rather in a set of 'shadow' variables held by each participant. Later, for an estimated system, there will still be one putative shared variable, but the local shadow copy may be less *precise*.

Decentralization depends on an additional distinction, a public namespace of *concepts* that may differ across agencies. An intuitive illustration is the difference between typing the address HTTP://WEATHER.ORG/LAX into a Web browser, and typing the concept "LA WEATHER REPORT" into a Web search engine's query box. The latter will return links to *many* different organizations' opinions of the weather in Los Angeles — but also to forecasts for Louisiana (which is also abbreviated LA).

3. New Consensus-Based Architectural Styles

Before we proceed to extend it, it behooves us to understand the properties that REST can induce on its own. We will then describe four different capabilities that can be added to REST: Asynchronous event notification; Routing messages through active proxies; Decision functions that select the current value of a shared resource; and Estimating current representations based upon past ones. Finally, we will combine all of these basic facilities to derive a new style for decentralized systems.

3.1Modern Web Architecture

There are many different network-based architectural styles, such as client-server and remote-data-access [11]. The style popularly known as "3-tier client/server" is a combination of both of those styles: presentation interface at a client, business logic on a server, and storage in a database. It was arguably the dominant style of application development until the mid-1990s, when it was eclipsed by architecture of the modern Web, as descibed by REpresentational State Transfer (REST, [13]).

In this style, software components are recast as network services. Clients request resources from servers (or proxy servers) using the resource's name and location, specified as a Uniform Resource Locator (URL, [3]). All interactions for obtaining a resource's representation are performed by synchronous request-response messages over an open, standard network protocol (HTTP/1.1, [12]). Requests can also be relayed via a series of proxies, filters, and caches, all without changing its semantics.

REST's essential distinction, though, is not found in such details. Rather, its layer of indirection[2] between abstract *resources* and concrete *representations* captured the Web's key insight for decentralizing hypertext — permitting *broken* links between independent sites [22].

Nevertheless, REST (and the Web, its archetypal application) still has significant limitations:

One-shot. Every request can only generate a single response; and if that response is an error message (or lost), there are no normative rules for proceeding.

One-to-one. Every request proceeds from one client to one server. Without a way to transfer information to a group of components, this leads to nested "proxy chains."

One-way. Every request must be initiated by a client, and every response must be generated immediately, precluding servers from sending asynchronous notifications. REST cannot model peer-to-peer relationships well.

3.2REpresentational State Transfer (REST)

To ground our exploration of these new issues, we began by restating REST to verify that it could induce the property of consensus. Our more rigorous correctness argument elucidated that REST depends on synchronized global clocks to ensure leases expire simultaneously.

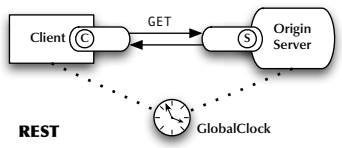

Figure 3: Illustration of the REST architectural style.

Synchronization still presumes that every response message specifies its lease interval. Many real-world ORIGINSERVERs do not specify when the next permissible resource update is scheduled. The external environment could update resources at random (e.g. editing a file by hand). One solution would be to set a default lease duration and force the server to discard or delay updates that arrive before that deadline, a *heartbeat* strategy.

[1] "A wire is just a renaming device for variables."
— Alain J. Martin, asynchronous VLSI designer [31]

[2] "Any problem in computer science can be solved by another level of indirection"— David Wheeler, chief prog., EDSAC [21]

Table 1: Summary of the REST style.

Goal	Refer to a centralized resource.
New Elements	GLOBALCLOCK makes explicit how clients, servers, and caches are synchronized.
New Constraints	ORIGINSERVER must always specify a consistent expiry deadline if the resource is ever to be updated.
Induced Property	*Consensus:* Ensures that local resource proxies ***could*** agree with leader's value. [polling would ensure it]

As an aside, we have also developed a variation, REST with Polling (REST+P) that is able to induce a weak form of simultaneous agreement. Using it, clients can follow leader values as long as the minimum lease interval is 3*d*.

3.3Asynchronous REST

To achieve simultaneous agreement as quickly as theoretically possible (within *d* of any change), we propose an event-based approach that permits the central resource to *broadcast* notifications of its state changes. Our insight is to recast the concept of a resource in REST as an event source that emits event notifications corresponding to each change in its representation(s).

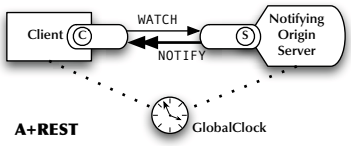

Figure 4: Illustration of the A+REST architectural style.

Rosenblum & Wolf [36] propose that "An *event* is the effect of the *termination* of an *invocation* of an *operation* on some *object of interest*…Events occur regardless of whether or not they're observed." Our stance is perhaps the opposite, insofar we consider that the very act of 'observation' to be what distinguishes events from messages. Specifically, our concern stems from the realization that 'on the wire,' there is little discernable difference between a messaging protocol and an event-notification protocol. However, there is a dramatic difference between the programming model for a batch message queue and an event handler. Thus, our view might be summarized as "*event notifications* are *messages* that cause *actions*."

In either case, though, there is a clear distinction between the occurrence of an abstract event and the concrete notification of an observation of one.[3]

As shown in Figure 4, a NOTIFYINGORIGINSERVER will transfer representations of *every* change to a resource as long as a client stays connected — a long-running WATCH request rather than a one-shot GET, with multiple NOTIFY replies. In practice, this is a significant implementation challenge across the public Internet [7].

Table 2: Summary of the A+REST style.

Goal	Refer to a changing centralized resource.
New Elements	NOTIFYINGORIGINSERVER that can send multiple responses to a WATCH request.
New Constraints	Every resource update must lead to transmission of a new representation to all watchers.
Induced Property	*Simultaneous Agreement:* Ensures that local resource proxies ***will*** agree with leader's value, even if it is being updated at ∞ Hz.

3.4Routed REST

While A+REST tackled the essential challenge of latency, message routing will focus on improving REST's support for multiple agencies.

The specific property we intend to induce is *multilateral extensibility:* the ability to add functionality to an application using components owned by several different agencies. The complication is that new 3rd and 4th parties may be trusted by the original client or server, yet distrust each other.

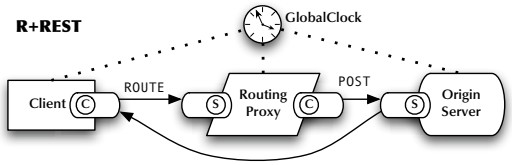

Figure 5: Illustration of the R+REST architectural style.

The reason this becomes a problem for REST is that, while it offers exemplary support for "active proxies" to add functionality without modifying deployed applications, it can only arrange them in linear proxy chains [24]. This permits intermediaries to tamper with messages.

Table 3: Summary of the R+REST style.

Goal	Compose services provided by multiple agencies.
New Elements	ROUTINGPROXY Component that permits clients to control relaying.
New Constraints	Every representation transfer must be justified by a corresponding edge in the web of trust.
Induced Property	*Multilateral Extensibility:* Can compose trusted invocations without requiring mutual trust.

With redirection of replies depicted in Figure 5, services can now be composed in a manner that eliminates unjustified trust relationships — and minimizes total latency as well.

[3] We are using the terms 'occurrence' and 'observation' as defined by the event lifecycle model of [35].

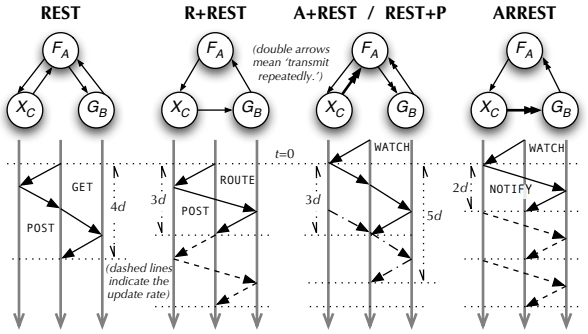

Figure 6: Latency of evaluating *F(G(X))* in several styles.

Composing multiple proxies, as in the simple chained evaluation of two functions owned by agencies *A* and *B* using data from a third, *C*. Figure 6 illustrates how the combination of Asynchrony and Routing leads to the highest performance: triangulation. It depicts five approaches to the problem of computing $F_A(G_B(X_C))$: two using *read()*, which could fail; and three using some form of *subscribe()*. In the case of REST+P, the dash-dot line shows the worst case. In order to compare characteristic update frequencies, we did not count the initial *subscribe()* requests, by assuming they occurred beforehand.

3.5REST with Delegation

The earliest efforts to extend the Web to support authoring immediately ran afoul of the "lost update" problem. Two editors using local, cached copies could easily overwrite each other's work using a simple PUT.

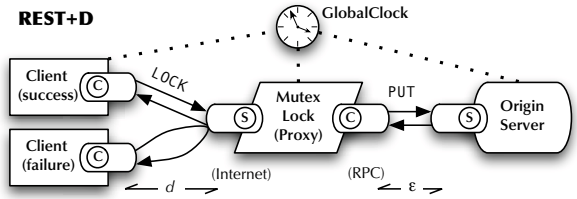

Figure 7: Illustration of the REST+D style.

The typical criteria for judging a distributed database are the 'ACID properties,' standing for Atomicity, Consistency, Isolation, and Durability [18]. Most practical systems have to make tradeoffs in the degree of 'ACIDity' to avoid tight coupling, though.

There is little an architectural style can do to enforce Durability (an implementation choice), or Consistency (an application-specific semantic). What REST+D *can* do is ensure total serialization of all updates to a resource.

The MUTEXLOCK Component is a proxy that wraps around an ORIGINSERVER to ensure mutually exclusive access. It contains an atomic test-and-set register identifying the only client whose request messages it will forward on to the ORIGINSERVER until instructed otherwise (or until the lock's lease expires). All other requests are simply discarded until the register is reset.

Table 4: Summary of the REST+D style.

Goal	Refer to a pairwise distributed read/write resource reliably.
New Elements	MUTEXLOCK Component ensures only one client at a time has write access to the origin server.
New Constraints	Lock must be acquired before attempting write. Current state of the resource must be read before attempting write.
Induced Property	*ACID (Pairwise) Simultaneous Agreement:* Clients can modify centralized resources within 3*d* — but only in the absence of contention.

The general form of the styles we derived that share control of a resource among several peers is to add a Decision function. Delegation is one such, akin to transaction processing monitors that temporarily centralized control at one location for a commit protocol. ARREST+D, a more distributed solution, eliminates the need for a single event router by adding a Distributed lock protocol, such as Lamport's Bakery algorithm [28].

4. New Consensus-Free Architectural Styles

Beyond the 'now horizon' or beyond an 'agency boundary' there is uncertainty referring to any remote resource. Latency and agency induce different sorts of uncertainty, though: loss of precision vs. loss of accuracy.

Communication between components is subject to network loss, delay, and congestion, all three of which increase message latency. Depending on the degree of auto-correlation a resource exhibits, relying on older information can reduce the precision of local estimates.

Furthermore, once resources are decentralized into an ensemble of independently controlled, local resources, such estimates also become less accurate. That is because there is no single 'true' value any more, since *facts* will be replaced by a host of agency-specific *opinions*.

Our insight for managing the risk of computing without consensus is a counterpoint to the ACID properties for centralized and distributed systems. Our so-called 'BASE' properties require decentralized systems to rely solely on Best-effort network messaging; to Approximate the current value of remote resources; to be Self-centered in deciding whether to trust other agencies' opinions; and Efficient when using network bandwidth.

4.1REST with Estimates

Our first step is to improve the *precision*: minimizing the error between the current value of the local resource

Table 5: Summary of the REST+E style

Goal	Refer to a read-only centralized resource beyond its now horizon
[Old] Elements	`[TCP/IP]`, `[CACHE]`, `[ACCESSCONTROL]`, `[CONTENTNEGOTIATION]`
New Constraints	Inertia assumes that the most recent representation is still valid, until cache revalidation fails.
Induced Property	*Approximate agreement:* The local proxy should be in simultaneous agreement *P%* of the time

and the (single) remote resource it corresponds to. However, to better understand the role of estimation in everyday usage of the Web, we first specified REST+E, an elaboration of REST's default behavior once $d=\infty$

Best-effort representation transfers are pushed down to the presentation layer of the network using TCP's sliding-window acknowledgement and retransmission protocols.

Approximate representations are returned by caches of several sorts: browser histories, caching proxies, and content distribution networks. Staleness is generally acceptable, even preferred in some cases.

Self-centered trust management is enforced by the use of server-based access controls, such as usernames and passwords.

Efficient representation formats are selected by client-driven content negotiation and compressed encodings.

4.2Putting it all together: ARRESTED

We combined each these four basic capabilities to derive new styles from REST for centralized (ARREST), distributed (ARREST+D), estimated (ARREST+E), and decentralized resources (ARRESTED).

By combining asynchrony and routing, we created first-class subscriptions, which are the building block for familiar event notification services. Architects can specify simultaneous notification to a group of components using multiple, persistent subscriptions.

With an event-based style, it is easier to induce both ACID (ARREST+D) and BASE (ARREST+E) properties. In particular, End-to-end Estimator functions can manage private proxy resources that replace references to shared resources. The upper half of Table 6 describes components that can store-and-forward retransmissions of lost or delayed notifications, predict future values from past information already received, discard information from untrusted sources, and summarize past data so as to send only the latest information. Ultimately, summarizers could even take advantage of excess bandwidth to reduce latency further by speculating about *future* states as well [39]. All of these extensions to REST serve to increase *precision* of an estimate of a single remote resource (ARREST+E).

The ultimate challenge we must address, though, is aid architects in designing applications that accurately model even uncertain information in decentralized systems. Our proposal for increasing accuracy is to assess multiple, simultaneously valid opinions of the same concept from several agencies. If we presume the errors of measurement are independent — and we are speaking of decentralizing some concept that is at least conceivably centralizable (a price, rather than an arbitrary value) — then multiple observations can reduce total error.

Of course, such a Decentralized decision function may be as simple as taking the minimum, maximum, mean, median or mode; or as complicated as a Turing-complete simulation of an underlying physical process. Any of these, though, is preferable to blocking while waiting for a lock from a transaction manager.

Table 6: Summary of the ARRESTED style.

Goal	New Elements	New Constraints	Induced Property
Refer to a read/write resource connected by a faulty network beyond its now horizon.	STOREANDFORWARD Connector that adds end-to-end retransmission & acknowledgement.	End-to-end retransmission of messages and acknowledgements.	*Best-Effort data transfer*: Cope with message loss.
	PREDICTOR Connector for encapsulating Turing-complete prediction functions.	Predict probable current state from past data (where possible).	*Approximate estimates*: Cope with message delay.
	TRUSTMANAGER Connector that drops notifications from untrusted sources.	Ensure that all reachable endpoints are also trusted.	*Self-Centered*: Cope with dynamic participation.
	SUMMARIZER Connector to resample queued events at lower frequency, reduce size.	Prohibit transmission of already-expired data. Eliminate buffering.	*Efficient data transfer*: Cope with net congestion.
Decentralize control of a shared resource across disjoint 'now horizons'	ASSESSOR Component that manages the risk of inter-agency disagreement over the 'true' value using a panel of opinions.	Eliminate reliable references to remote resources; only contingent assessments remain.	*Consensus-freedom:* avoid presuming feasibility of consensus.

5. Implementation Experience

Deriving new architectural styles and validating the properties they can induce by construction is a conceptual exercise. It is at least as important to establish that all of the new components, connectors, and constraints are actually implementable; and that actual applications can be constructed in each style.

5.1Infrastructure

We have over five years of experience developing an experimental event-routing infrastructure along these lines. Eventually, that laid the foundation for the award-winning commercial edition sold by KnowNow, Inc [40]. The original prototype implementations were released separately as an open-source project in December 2002 called MOD_PUBSUB (by analogy to the Apache MOD_* naming convention for Web server extensions) [25]. Using MOD_PUBSUB with existing web tools and browsers, web pages can now respond to incoming events from the

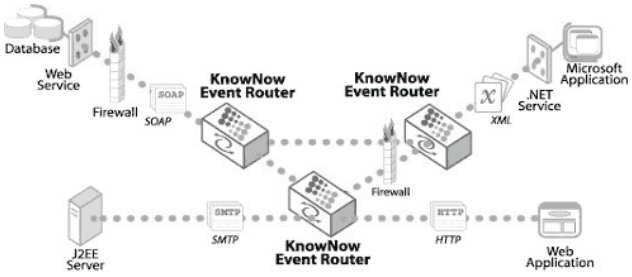

Figure 8: A commercial example of ARREST infrastructure.

network — in real-time; with plain text, HTML, XML, and/or SOAP; and without relying on Java or ActiveX. It works with Mozilla, Netscape, and Microsoft browsers.

Both event notification services implement many of the component types introduced in §3 and §4, such as NOTIFYINGORIGINSERVER. Others are application-specific enough that they must be supplied by the architect, such as ESTIMATORs and ASSESSORs. Yet others appear as sample applications and reusable libraries, such as an implementation of STOREANDFORWARD called ZACK.

5.2Applications

We have focused on a specific application to illustrate our development methodology and claims for our styles: AUTOMARKET, a used-car auction shown in Figure 9.

Centralized. In the simplest mode, the owner of the central event router (web server) is the only agent that can set prices. We tested this by writing 'classified ad' events directly into files on the server's disk (no remote access).

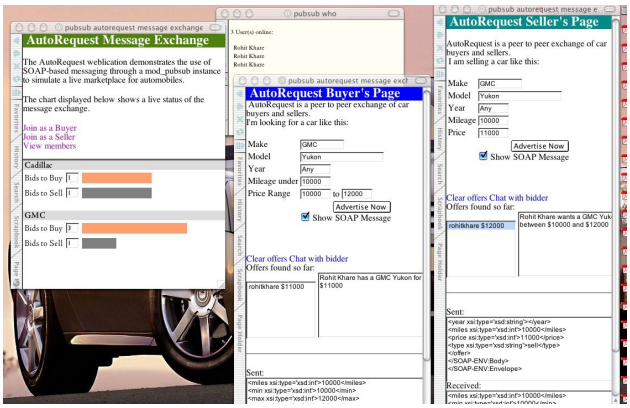

Figure 9: Screenshot of our used car auction example.

Distributed. The next step is allowing multiple bidders to publish new bids to a shared topic. We tested this with both REST+D by relying on the central router to arbitrate the order bids arrived in and to guarantee delivery; and with ARREST+D by using our ZACK protocol to count acknowledgements from other traders.

Estimated. Since our example is not particularly high-frequency, the primary type of latency risk is disconnection rather than a few seconds' arbitrage. Per REST+E, AUTOMARKET defaults to a policy of inertia by displaying any recent bid within the last 24 hours. To experiment with ARREST+E, we connected the same feed to KnowNow's Excel spreadsheet adaptor and used its built-in time-series data to extrapolate current prices.

Decentralized. Finally, we adapted our user interface to work with generic concepts such as "truck prices." Rather than presenting price data for several different vehicles, we let users configure which types they considered to be "trucks" and calculated the average current price. Note that this did not rely upon bidders to classify what vehicles were trucks, nor upon "hidden hierarchy" in the router's topic names, but could be set individually by each trader.

5.3Methodology

Based on our experience writing decentralized applications, we can identify a few common issues that arose.

Since ARRESTED is an event-based architectural style, the first phase of developing a decentralized application is still generic event-based analysis: identifying the components, event sources, subscription qualifiers, message formats, and the like. The second phase is a methodology specifically addressing unique concerns raised by decentralization. It has five steps:

1. Identify the agencies. In any real marketplace, the interests of traders, brokers, and the exchange diverge significantly. Note that a single organization may develop the software used to enact all of these agencies' roles, but

the design must remain robust in the face of independent implementations. Part of the challenge of developing architectural styles for decentralization is coming up with abstract models of software written by others.

2. Characterize the latencies. The next step is to characterize the latencies, both of the networks the application will run on top of, and of the real-world phenomena that it is attempting to represent. The latter may be more challenging: it may appear that the NYSE's ticker stops updating a stock's price on weekends, but the "after-hours" valuation may keep changing as news breaks.

3. Establish the web of trust. This requires more than merely authenticating credentials for each agency; it also determines which external resources ought to be considered "equivalent" to the same concept. This could range from lexical matching to 'Semantic Web' tags [2].

4. Eliminate remote references. This is the constraint unique to the ARRESTED architectural style: replacing all resources owned by external agencies with private assessments. Architects face tradeoffs between different prediction engines, compression engines, and other types of estimators; our style at least isolates such complexity.

5. Track the provenance of events. In an era of profligate computing resources, event notification is an appropriate use of surplus bandwidth, and audit trails are an appropriate use of surplus storage. Ideally, every datum displayed by a user interface ought to indicate its confidence interval. Furthermore, tracing ownership is essential to re-evaluate results after security breaches invalidate data or rollback/"anti-messages" arrive [20].

6. Conclusions

In this paper, we presented: a formal definition of decentralization; an analysis of the limitations of consensus-based software architectural styles; derivation of new architectural styles that can enforce the required properties; and implementations that demonstrate the feasibility of those styles and sample applications.

Figure 10 summarizes our findings. First, we identified two basic factors limiting the feasibility of consensus: latency and agency. These correspond to two boundaries, indicated by dashed lines: the 'now horizon' within which components can refer to the value of a variable 'right now'; and an agency boundary within which components can trust each other. Another way to describe them is that the now horizon separates consensus-based styles from consensus-free ones; and the agency boundary separates master/slave styles from peer-to-peer ones.

First, we identified four new capabilities that could be combined with REST individually to induce the properties we desired: events, routes, locks, and estimates. Then, we were able to combine these to derive four new styles optimized for each of the four types of resources.

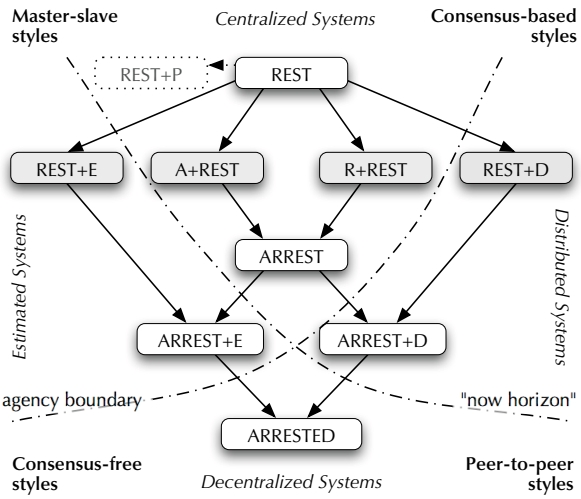

Figure 10: Diagram summarizing our four new architectural styles, derived from four capabilities added to REST.

For centralized resources, we enforce simultaneous agreement by extending REST into an event-based architectural style by adding Asynchronous event notification and Routing through active proxies (ARREST). For distributed control of shared resources, we enforce ACID transactions by further extending REST with end-to-end Decision functions that enable each component to serialize all updates (ARREST+D).

The alternative to simultaneous agreement is *decentralization*: permitting independent agents to make their own decisions. This requires accommodating four intrinsic sources of uncertainty that arise when communicating with remote agencies: loss, congestion, delay, and disagreement. Their corresponding constraints are Best-effort data transfer, Efficient summarization of data to be sent, Approximate estimates of current values from data already received, and Self-centered trust management.

These so-called 'BASE' properties can be enforced by replacing references to shared resources with end-to-end Estimator functions. Such extensions to REST can increase *precision* of measurements of a single remote resource (ARREST+E); as well as increase *accuracy* by assessing the opinions of several different agencies (ARRESTED) to eliminate independent sources of error.

Furthermore, application of these styles to real-world problems has been shown to be both feasible and effective, using both open-source and commercial tools.

Acknowledgements

This work is based on the first author's doctoral dissertation, which also benefited from the support of Dr. André van der Hoek and Dr. Debra J. Richardson. The authors are also grateful for the assistance of Dr. Joseph Touch, Dr. E. James Whitehead, Dr. Roy T. Fielding, Dr. Peyman Oreizy, Eric M. Dashofy, and Adam Rifkin.

This material is based upon work supported by the National Science Foundation under Grant #0205724.

References

[1] Association for Computing Machinery. *ACM Computing Classification System*. 1998. http://www.acm.org/class/1998/ccs98.html

[2] Berners-Lee, T., Hendler, J. and Lassila, O. *The Semantic Web* in *Scientific American*, May, 2001. vol. *284* (5), pp. 34-43.

[3] Berners-Lee, T., Masinter, L. and McCahill, M. *RFC 1738: Uniform Resource Locators (URL)*. Internet Engineering Task Force, December 1994.

[4] Birman, K. P. and Joseph, T. A. *Exploiting Virtual Synchrony in Distributed Systems*, in *Eleventh Symposium on Operating Systems Principles*, (Austin, Texas, 1987).

[5] Borbeley, A.-M. and Kreider, J. F. (eds.). *Distributed Generation: The Power Paradigm for the New Millenium* CRC Press, Boca Raton, Florida, 2001

[6] Brewer, E. *Invariant Boundaries (Invited Keynote)*, in *9th International Workshop on High Performance Transaction Systems (HPTS)* (Pacific Grove, CA, October 14-17 2001). http://research.microsoft.com/~jamesrh/hpts2001/presentations/hpts2001-brewer.ppt

[7] Carzaniga, A., Rosenblum, D. S. and Wolf, A. L. *Design and Evaluation of a Wide-Area Event Notification Service* in *ACM Transactions on Computer Systems*, 2001, *9* (3). pp. 332-383.

[8] Chon, K. *Information Processing in Electricity Distribution Systems*, in *Proceedings of the Annual Conference (Volume 2)*, (1978), ACM Press, pp. 979-984.

[9] Curtiss, P. *Control of Distributed Electrical Generation Systems* in *ASHRAE Transactions*, 2000, *106* (1).

[10] Dickerson, C. *The Battle for Decentralization* in *Infoworld*, May 2, 2003. http://www.infoworld.com/article/03/05/02/18OPconnection_1.html

[11] Fielding, R. *Architectural Styles and the Design of Network-based Software Architectures* (PhD Thesis), University of California, Irvine, Information and Computer Science, Irvine, CA, 2000.

[12] Fielding, R., Gettys, J., Mogul, J. C., Frystyk, H., Masinter, L., Leach, P. and Berners-Lee, T. *RFC 2616: Hypertext Transfer Protocol – HTTP/1.1*. Internet Engineering Task Force, June 1999.

[13] Fielding, R. T. and Taylor, R. N. *Principled Design of the Modern Web Architecture* in *ACM Transactions on Internet Technology (TOIT)*, 2002, *2* (2). pp. 115-150.

[14] Fisher, M. J., Lynch, N. A. and Paterson, M. S. *Impossibility of Distributed Consensus with One Faulty Process* in *Journal of the ACM*, 1985, *32* (2). pp. 374-382.

[15] Fox, A. and Brewer, E. *Harvest, Yield, and Scalable Tolerant Systems*, in *Proceedings HotOS-VII*, (1999). http://swig.stanford.edu/pub/publications/harvest_yield.pdf

[16] Giesen, J., Wattenhofer, R. and Zollinger, A. *Towards a Theory of Peer-to-Peer Computability*, in *Proceedings of the 9th International Colloquium on Structural Information and Communication (SIROCCO)*, (Andros, Greece, June 2002). http://distcomp.ethz.ch/projects/p2p.html

[17] Gilbert, S. and Lynch, N. A. *Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-tolerant Web Services* in *SIGACT News*, 2002, *33* (2). http://theory.lcs.mit.edu/tds/papers/Gilbert/Brewer6.ps

[18] Gray, J. and Reuter, A. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, San Francisco, CA, 1992. 1070pp.

[19] Halstead, R. *Multilisp :A Language for Concurrent Symbolic Computation* in *ACM Transactions on Programming Languages and Systems*, 1985, *7* (4). pp. 501-538.

[20] Jefferson, D. R. *Virtual time* in *ACM Transactions on Programming Languages and Systems*, 1985, *7* (3). pp. 404-425.

[21] Khare, R. What's in a Name? Trust. (Internet-Scale Namespaces, Part II) in IEEE Internet Computing, 1999, 3 (6). pp. 80-84.

[22] Khare, R. *Who Killed Gopher? An Extensible Murder Mystery* in *IEEE Internet Computing*, Jan/Feb, 1999. vol. *3* (1), pp. 81-84. http://www.ics.uci.edu/~rohit/IEEE-L7-http-gopher.html

[23] Khare, R. *Extending the Representational State Transfer (REST) Architectural Style for Decentralized Systems* (PhD Thesis), University of California, Irvine, Information and Computer Science, Irvine, CA, 2003.

[24] Khare, R. and Rifkin, A. *Composing Active Proxies to Extend the Web*, in *OMG-DARPA-MCC Workshop on Compositional Software Architectures*, (Monterey, CA, January 1998 1998).

[25] Khare, R., Rifkin, A., Sitaker, K. and Sittler, B. *mod_pubsub: an open-source event router for Apache*, 2002. http://mod-pubsub.sourceforge.net/

[26] Kubiatowicz, J. *Extracting Guarantees from Chaos* in *Communications ACM*, 2003, *46* (2). pp. 33-38.

[27] Lamport, L. *Time, Clocks and the Ordering of Events in a Distributed System* in *Communications of the ACM*, 1978, *21* (7). pp. 558-565.

[28] Lamport, L. *The Mutual Exclusion Problem: Part II-Statement and Solutions* in *Journal of the ACM*, 1986, *33* (2). pp. 327-348.

[29] Lamport, L., Shostak, R. and Pease, M. *The Byzantine Generals Problem* in *ACM Transactions on Programming Languages and Systems*, 1982, *4* (3). pp. 382-401.

[30] Lynch, N. A. *Distributed Algorithms*. Morgan Kaufmann, San Mateo, CA, 1996. 904pp.

[31] Martin, A. *Wires, Forks, and Multiple-Output Gates* in Gries, D. ed. *Beauty is Our Business*, 1990, p. 304.

[32] Milojicic, D. S., Kalogeraki, V., Lukose, R., Nagaraja, K., Pruyne, J., Richard, B., Rollins, S. and Xu, Z. *Peer-to-Peer Computing*. HP Labs, Palo Alto, CA, 2002. 51pp.

[33] Oram, A., Minar, N. and Shirky, C. *Peer-to-Peer: Harnessing the Power of Disruptive Technologies* 1st ed. O'Reilly & Associates, 2001. 432pp.

[34] Renesse, R. v., Birman, K. P. and Maffeis, S. *HORUS: A Flexible Group Communication System* in *Communications of the ACM*, 1996, *39* (4). pp. 76-83.

[35] Rosenblum, D. S. and Wolf, A. L. *A Design Framework for Internet-Scale Event Observation and Notification*, in *6th European Software Engineering Conference/5th ACM SIGSOFT Symposium on the Foundations of Software Engineering*, (Zurich, Switzerland, September 1997), Springer-Verlag, pp. 344-360.

[36] Rosenblum, D. S. and Wolf, A. L. *Internet Scale Event Notification*, in *Workshop on Internet Scale Event Notification (WISEN'98)*, (Irvine, CA, July 13-14 1998). `http://www.ics.uci.edu/~irus/wisen/wisen98/abstracts/abs_rosenblum.html`

[37] Schneier, B. *Secrets and Lies*. John Wiley & Sons, Inc., 2000. 432pp.

[38] Thornley, J. *A Parallel Programming Model with Sequential Semantics* (PhD Thesis), California Institute of Technology, Computer Science Department, 1996.

[39] Touch, J. D. *Mirage: A Model for Latency in Communication* (PhD Thesis), University of Pennsylvania, Computer and Information Science, 1992.

[40] Udell, J., Gillmor, S. and Eds. *2002 Technology of the Year Award: Publish/Subscribe Technology: KnowNow 1.5* in *Infoworld*, January 24, 2003. `http://www.infoworld.com/article/03/01/24/2002TOY-sb_1.html`