Remote Belief: Preserving Volition for Loosely Coupled Processes

Nuh Aydin
Department of Mathematics
Kenyon College
Gambier, OH USA 43022-9623
aydinn@kenyon.edu

Paolo A. G. Sivilotti
Department of Computer and Information Science
The Ohio State University
Columbus, OH USA 43210-1277
paolo@cis.ohio-state.edu

Abstract

Knowledge has proven to be a useful and fundamental formalism for reasoning about distributed systems. The application of this formalism, however, entails a loss of volition on the part of processes about which something is known. This loss of volition is often not appropriate in loosely coupled distributed systems. In this paper, we generalize the formal characterization of knowledge into one of belief. *Belief has the advantage of allowing processes to maintain volition. We examine some of the similarities and surprising differences between knowledge and belief. We also present some examples of distributed applications that are more conveniently characterized with belief rather than knowledge.*

1. Introduction

The general concepts of knowledge and belief have received considerable attention in such diverse fields as philosophy [12], artificial intelligence [6, 15] game theory [1], economics [17], linguistics [3], and computer science [2, 11, 5]. Although the models and contexts are not always the same, there are considerable similarities among them. A survey by Halpern [9] describes some of the common threads in the area of knowledge in various fields. In the context of distributed systems, knowledge has been defined in terms of *isomorphisms* of *computations* with respect to processes [2], in terms of global and local *states* being *indistinguishable* with respect to an agent [6], and in terms of *equivalence* of *scenarios* [13, 10].

A fundamental result in the application of knowledge to distributed systems is that a process can only gain knowledge about remote predicates by receiving a message and conversely can only lose knowledge by sending a message. That is, if a process P knows something about the state of a remote process Q, Q can only change that part of its state if P sends a message. Q has lost volition, that is, the autonomy to change this part of its local state. This loss of autonomy, however, may be too strict a requirement for some distributed applications (*e.g.*, *ad hoc* networks) or some aspects of local state (*e.g.*, failure modes which are controlled by the environment).

In this paper, we generalize the characterization of knowledge in distributed systems into one of *belief*. This new formalism is based on isomorphisms of system computations. Unlike knowledge, belief allows processes to maintain autonomy in the control of their local state. Belief about remote predicates can be gained or lost by either sending or receiving messages or by internal events.

The rest of this paper is organized as follows. In Section 2, we review the formal definition of knowledge and the fundamental theorems for its application to distributed systems. In Section 3, we formally define the notion of belief and establish knowledge as a special case. We then investigate some similarities and surprising differences between the two concepts. Sections 4 and 5 give the basic theorems for working with belief and describe how it can be gained, lost, and transferred. We illustrate the application of belief by examining two common constructs in distributed systems: directory services and asynchronous leases. Both constructs are more suitably characterized in terms of belief rather than knowledge. Section 7 concludes and indicates directions for future work.

2. Background: Knowledge in Distributed Systems

2.1. Computations

An *event* on a process is either a send, a receive, or an internal event. A *process computation* is a finite sequence of events on that process. A process is characterized by a set of process computations. This set is prefix closed. Let z be any sequence of events in a distributed system. The projection of z on a component process p, denoted by z_p, is the subsequence of z consisting of all events on p. A finite sequence

of events z is a *system computation* of a distributed system if: (i) for all processes p, z_p is a process computation, and (ii) for every receive event in z, there is the corresponding send event in z.

System computations are also prefix closed. The set of all system computations is denoted by X, while a particular system computation is denoted by x, y, or z. The concatenation of two sequences y and z is written $(y; z)$. For sequences y and z, $y \leq z$ denotes that y is a prefix of z; in that case (y, z) denotes the suffix of z obtained by removing y from z. The set of all processes in the system is denoted by D and for any process set P, $\bar{P} = D - P$. For a predicate b defined on X, b **at** x denotes its value for computation x. We assume that all predicates are total and we assume that for a given system there are only finitely many possible events on any of the processes.

2.2. Isomorphism and Knowledge

For a process p, the relation $[p]$ on the set of all system computations is defined as follows.

Definition 2.1. For $x, y \in X$, $x[p]y$ if $x_p = y_p$.

In words, two computations x and y are isomorphic with respect to p if p's computation is the same in x and y. That is, the computations x and y cannot be distinguished by p.

In general, for a process set P, the relation $[P]$ is defined as

Definition 2.2. For $x, y \in X$, $x[P]y$ if for all $p \in P$, $x[p]y$.

That is, $x[P]y$ means that the computations x and y cannot be distinguished by any of the processes in P.

The relation $[P]$ is an equivalence relation on X. We denote the equivalence class of $[P]$ containing x by $[P]_x$ (*i*.e., $[P]_x = \{y \in X : y[P]x\}$). For a set A, $|A|$ stands for its cardinality. For a computation x, $|x|$ is its length.

Definition 2.3. Let $n > 0$ and P_i be process sets, $0 \leq i \leq n$.
$x[P_0 \ldots P_n]z$ means $x[P_0 \ldots P_{n-1}]y$ and $y[P_n]z$, for some computation y.

Principle of computation extension: Let e be an event on P.

1. If e is an internal or send event, and $(x; e)$ is a computation, and $x[P]y$, then $(y; e)$ is a computation.
2. If e is an internal or receive event and $(x; e)[P]y$, then $(y - e)$ is a computation, where $(y - e)$ is the sequence obtained by deleting e from y.

Definition 2.4 (Knowledge). $(P$ **knows** $b)$ **at** x if for all $y \in [P]_x$, b **at** y is true.

2.3. Local Predicates

Let b be a predicate on system computations, and P be a set of processes. The predicate P **sure** b is defined as follows.

Definition 2.5. $(P$ **sure** $b)$ **at** x if $(P$ **knows** $b)$ **at** x or $(P$ **knows** $\sim b)$ **at** x.

In other words, $(P$ **sure** $b)$ **at** x means that P knows the value of b after x.

Definition 2.6. b is local to P if, for all x, $(P$ **sure** $b)$ **at** x.

That is, the value of b is always known to P. Local predicates capture our intuitive notion of a predicate whose value is controlled by the actions of the process to which it is local.

Notation. The operator **knows** has higher binding than **at**, so $(P$ **knows** $b)$ **at** x may be written as P **knows** b **at** x.

3. Definition of Belief

Knowledge has proven to be useful in establishing lower bounds for some distributed algorithms [2]. In general, however, many algorithms cannot be characterized with knowledge because of its strict requirements. For example, one property of knowledge is that only true things can be known (P **knows** b **at** $x \implies b$ **at** x). Consider a shared network printer. If a client comes to know that the printer has paper in its tray, then it must be the case that there is indeed paper in its tray. Another consequence of the classic definition of knowledge is that knowledge about a remote predicate cannot be lost by receiving a message. In the case of the shared printer, a client cannot "forget" that there is paper in the printer tray simply by receiving messages. Knowledge requires that the client *send* a message in order for the remote predicate to become false. In practical terms, this means the printer cannot empty its paper tray unless all its clients (that knew the tray was nonempty) send a message with their permission! Until that happens, the printer must maintain paper in its paper tray.

Clearly, this is too restrictive a requirement for loosely coupled systems, where clients can dynamically join or leave, and where client crashes or suspensions are frequent. In practice, the printer *is* allowed to modify its local state (*i.e.*, empty its paper tray). That is, the state of the paper tray is never *known* to remote clients in the formal sense.

Nevertheless, messages indicating that there is paper in the tray can be useful to clients. While, strictly speaking, receiving such a message does not increase the knowledge a client has about the current state of the printer tray, it clearly communicates something about a recent past state,

with the corresponding likely consequences for the current state. With this in mind, we define a notion of *belief* for distributed systems.

Let $[P_b]_x$ denote the subset of $[P]_x$ where predicate b holds true. That is, $[P_b]_x = \{y \in [P]_x : b \textbf{ at } y \text{ is true}\}$. Then, P **knows** b **at** x is equivalent to $[P]_x = [P_b]_x$. A natural way to generalize knowledge is the following: process P believes b at x means b holds for most (or some) computations isomorphic to x with respect to P. More precisely, we might quantify belief by attaching a real number α with $0 \leq \alpha \leq 1$. So, a first attempt to define "P believes b at x with α" would be

$$\frac{|[P_b]_x|}{|[P]_x|} \geq \alpha$$

However, there is a technical problem with this definition. Since the computations can be arbitrarily long, the set $[P]_x$ can be infinite, as can the numerator of the above expression. (If the numerator is finite and the denominator is infinite, then the fraction would be 0). Therefore, we need to define the belief more carefully. To this end, we first introduce some notation.

For a natural number N, let $X_N = \{x \in X : |x| \leq N\}$. That is, X_N is the set of all system computations of length at most N. If there are n processes in the system and each process has k possible actions, then $|X_N| \leq \sum_{i=0}^{N} (kn)^i = ((kn)^{N+1} - 1)/(kn - 1)$. For each computation x, process set P, predicate b, and natural number N such that $X_N \cap [P]_x$ is not empty, we define a (rational) number $a_N = a_N(P, b, x)$ by

$$a_N = \frac{|X_N \cap [P_b]_x|}{|X_N \cap [P]_x|}$$

We obtain a sequence of rational numbers $(a_N)_{N \geq 0}$ with $0 \leq a_N \leq 1$. Note that a_N represents the fraction of the computations of length at most N in the class $[P]_x$ for which b is true. A second attempt to define belief would be to use the limit of this sequence. That is, P believes b at x with α when $\lim_{N\to\infty} a_N \geq \alpha$. Unfortunately, there is still a technical problem with this attempt: Not every sequence of real (or rational) numbers has a limit. However, every sequence of real numbers does have a lower limit (denoted by $\liminf$) and an upper limit (denoted by $\limsup$). The lower (upper) limit of an infinite sequence S is the infimum (supremum) of the limits of all infinite subsequences of S. There is no reason to prefer one of these limits over the other in defining belief. Indeed, the average of the two limits has some pleasing symmetries.

Definition 3.1 (Belief). P **bel**$_\alpha$ b **at** x if

$$\frac{1}{2}\liminf_{N\to\infty} a_N + \frac{1}{2}\limsup_{N\to\infty} a_N \geq \alpha$$

Notation: We will often write $\lim a_N$, $\liminf a_N$, and $\limsup a_N$ to mean $\lim_{N\to\infty} a_N$, $\liminf_{N\to\infty} a_N$ and $\limsup_{N\to\infty} a_N$ respectively.

Although $\lim a_N$ need not exist for an arbitrary sequence a_N, $\liminf a_N$ and $\limsup a_N$ always do. When $\lim a_N$ does exist, $\lim a_N = \liminf a_N = \limsup a_N = \frac{1}{2}\liminf a_N + \frac{1}{2}\limsup a_N$. Moreover, since $0 \leq a_N \leq 1$ for all N, both $\liminf a_N$ and $\limsup a_N$ are between 0 and 1.

4. Properties of Belief

Our first observation is that P **knows** b implies P **bel**$_\alpha$ b for any $0 \leq \alpha \leq 1$; in particular, P **bel**$_1$ b. Conversely, when the sequence a_N is constant, P **bel**$_1$ b implies P **knows** b. In general, however, a belief of 1 is not the same as knowledge. This distinction is illustrated by the following example.

Example 4.1. Consider the system consisting of two processes, p and q. Let p's and q's computations consist of ${s_p}^*$ and ${s_q}^*$ respectively, where s_p and s_q are internal events (skip actions) on the corresponding processes. The set of system computations is $(s_p|s_q)^*$. Let x be $s_q s_q$ and let b be the predicate "There are at least 2 events". It is not the case that p **knows** b **at** x (since $y[p]x$ where $y = s_q$, yet $\sim b$ **at** y). However, it *is* the case that p **bel**$_1$ b **at** x (since $a_N = \frac{N-1}{N+1}$ for $N \geq 1$, and hence $\lim_{N\to\infty} a_N = 1$).

The following lemma states some basic properties of belief.

Lemma 4.1.

1. *If* $\beta > \frac{1}{2}(\liminf a_N + \limsup a_N)$ *then* $\sim$ (P **bel**$_\beta$ b **at** x)
2. P **bel**$_\alpha$ b *implies* P **bel**$_\beta$ b *for all* $\beta \leq \alpha$
3. (P **bel**$_\alpha$ b)$\vee \sim$ (P **bel**$_\alpha$ b)
4. P **bel**$_0$ b *for any predicate* b.
5. (P **knows** b) $\Rightarrow$ (P **bel**$_1$ b)
6. *If* a_N *is constant,* (P **bel**$_1$ b) $\Rightarrow$ (P **knows** b)
7. *If there exists an integer* N_0 *such that* $|x| \leq N_0$ *for all* $x \in X$*, that is if the lengths of computations are bounded, then*
   i) $a_N = a_M$ *for all* $N, M \geq N_0$*, that is the sequence* a_N *eventually becomes constant.*
   ii) (P **bel**$_1$ b) $\Leftrightarrow$ (P **knows** b)
8. *If* $x[P]y$ *then* (P **bel**$_\alpha$ b **at** x) $\Leftrightarrow$ (P **bel**$_\alpha$ b **at** y)
9. $[(P$ **bel**$_\alpha$ $b) \wedge (b \Rightarrow b')] \Rightarrow (P$ **bel**$_\alpha$ $b')$

10. $((P \textbf{ bel}_\alpha\ b) \wedge (P \textbf{ bel}_\beta\ b')) \Rightarrow P \textbf{ bel}_{\max\{0,\alpha+\beta-1\}}\ (b \wedge b')$

11. $((P \textbf{ bel}_\alpha\ b) \wedge (P \textbf{ bel}_\beta\ b')) \Rightarrow P \textbf{ bel}_{\max\{\alpha,\beta\}}\ (b \vee b')$

12. $((P \textbf{ bel}_\alpha\ b) \vee (P \textbf{ bel}_\beta\ b')) \Rightarrow P \textbf{ bel}_{\min\{\alpha,\beta\}}\ (b \vee b')$

13. $(P \textbf{ bel}_\alpha\ b \textbf{ at } x) \Rightarrow (P \textbf{ bel}_{1-\alpha}\ \sim b)$ *for the maximal belief level of* α, i.e., *for* $\alpha = \frac{1}{2}(\liminf a_N + \limsup a_N)$

14. $P \textbf{ bel}_\alpha\ (Q \textbf{ knows } b) \Rightarrow P \textbf{ bel}_\alpha\ b$

15. $P \textbf{ bel}_\alpha\ (P \textbf{ bel}_\beta\ b) \Rightarrow P \textbf{ bel}_\beta\ b$, *if* $\alpha > 0$

16. $([P \cup Q]_x \subseteq [P_b]_x \cup [Q_b]_x) \Rightarrow P \cup Q \textbf{ knows } b \textbf{ at } x$

Proof. Most of these properties follow from the definition immediately. We give a proof of 13 as an example. Let $\tilde{a}_N$ be the sequence associated with $\sim b$, so that $a_N + \tilde{a}_N = 1$ for all N. Therefore, $\tilde{a}_N = 1 - a_N$ and $\liminf(\tilde{a}_N) = \liminf(1-a_N) = 1+\liminf(-a_N) = 1-\limsup(a_N)$ and $\limsup(\tilde{a}_N) = \limsup(1 - a_N) = 1 + \limsup(-a_N) = 1 - \liminf(a_N)$. Hence, $\frac{1}{2}(\liminf \tilde{a}_N + \limsup \tilde{a}_N) = \frac{1}{2}(2 - (\liminf a_N + \limsup a_N)) = 1 - \frac{1}{2}(\liminf a_N + \limsup a_N) = 1 - \alpha$. □

The following lemma is proven using Theorem 4 in [2].

Lemma 4.2. *For arbitrary process sets* $P_1 \ldots P_n$, *predicate* b, *and computations* x *and* y, *if* $(P_1 \textbf{ knows } \ldots P_{n-1} \textbf{ knows } (P_n \textbf{ bel}_\alpha\ b \textbf{ at } x))$ *and* $x[P_1 \ldots P_n]y$, *then* $(P_n \textbf{ bel}_\alpha\ b \textbf{ at } y)$.

Proof. Assume that $(P_1 \textbf{ knows } \ldots P_{n-1} \textbf{ knows } (P_n \textbf{ bel}_\alpha\ b \textbf{ at } x))$ and $x[P_1 \ldots P_n]y$. We need to show $(P_n \textbf{ bel}_\alpha\ b \textbf{ at } y)$. Since $x[P_1 \ldots P_n]y$, there exists a computation z such that $x[P_1 \ldots P_{n-1}]z$ and $z[P_n]y$. By Theorem 4 in [2], we have P_{n-1} **knows** $(P_n \textbf{ bel}_\alpha\ b)$ **at** z which implies $P_n \textbf{ bel}_\alpha\ b \textbf{ at } z$. Finally, since $z[P_n]y$, $P_n \textbf{ bel}_\alpha\ b \textbf{ at } y$. □

5. Transfer of Belief

The following lemma from [2] is of fundamental importance in transfer of knowledge. We state it below and give an alternative and more elementary proof. It gives strong restrictions on the transfer of knowledge about remote predicates. We show that this lemma does not hold for belief. Therefore, the notion of belief does not have these restrictions, making it more flexible for systems where processes are loosely coupled.

Lemma 5.1 (Knowledge Transfer). *Let* b *be a predicate that is local to* $\bar{P}$ *and* $(x; e)$ *be a computation where* e *is an event on* P.

1. *If* e *is a receive, then* $(P \textbf{ knows } b \textbf{ at } x) \Rightarrow (P \textbf{ knows } b \textbf{ at } (x; e))$ *(Knowledge is not lost).*
2. *If* e *is a send, then* $(P \textbf{ knows } b \textbf{ at } (x; e)) \Rightarrow (P \textbf{ knows } b \textbf{ at } x)$ *(Knowledge is not gained).*
3. *If* e *is an internal event, then* $P \textbf{ knows } b \textbf{ at } x = (P \textbf{ knows } b \textbf{ at } (x; e))$ *(Knowledge is unchanged).*

Proof. We prove Part 1. Let z be a computation such that $(x; e)[P]z$. Consider $(z - e)$, which is a computation by the principle of computation extension. We have $(z - e)[P]x$, and therefore $b \textbf{ at } (z - e)$ and $b \textbf{ at } z$ (since b is local to $\bar{P}$). □

Unlike knowledge, a process's belief about remote predicates may change by any one of the three types of actions (events). Indeed, a receive action or a send action can increase or decrease belief. Perhaps more surprisingly, even an internal event in a process may change its belief about remote predicates.

Lemma 5.2 (Belief Transfer). *Let* b *be a predicate that is local to* $\bar{P}$ *and let* x *and* $(x; e)$ *be computations where* e *is an event on* P. P*'s belief of* b *from* x *to* $(x; e)$ *may increase or decrease, regardless of whether* e *is a receive, a send, or an internal action.*

The following sequence of examples illustrates some of these possibilities. The examples consider a system of two processes, p and q. In each example, p has some belief regarding a predicate b that is local to q. The first example shows how a receive action can increase or decrease belief, depending on the value that is received.

Example 5.1. Let the events on q consist of three possible internal events (s_q, h, and t) and two possible send events (S_h and S_t). Process computations for q are strings of the form $hs_q^*S_hs_q^*$ or $ts_q^*S_ts_q^*$. Informally, q begins by flipping a coin and then eventually sends the result of the coin toss to p. The events on p consist of the internal event s_p and the receive events R_h and R_t. Process computations for p are strings of the form $s_p^*R_h$ or $s_p^*R_t$.

Now let b be the predicate "an h occurred" and x be the computation hS_h. Then we have $a_N = \frac{\frac{1}{2}N(N+1)}{N(N+1)+1}$, and hence $p \textbf{ bel}_{\frac{1}{2}}\ b \textbf{ at } x$.

Once p receives a message, however, this belief changes. Extending x with R_h we obtain $p \textbf{ bel}_1\ b \textbf{ at } (x; R_h)$. In fact, we have $P \textbf{ knows } b \textbf{ at } (x; R_h)$. On the other hand, extending x with R_t yields $p \textbf{ bel}_0\ b \textbf{ at } (x; R_t)$. Thus, while receive actions can only increase knowledge, they can increase or decrease belief.

The next example illustrates how sending a message can increase belief in a remote predicate.

Example 5.2. Let process computations for q be strings of the form $R_hhs_q^*$ or $R_tts_q^*$. Informally, q waits to receive a message from p and then generates the corresponding event.

Process computations for p are strings of the form $s_p^* S_h$ or $s_p^* S_t$.

Again, b is the predicate "an h occurred" and let x be the empty computation. Then we have P **bel**$_0$ b **at** x. Extension of this computation with a send by p, however, can strictly increase p's belief: P **bel**$_1$ b **at** $(x; S_h)$.

The next example shows that modifications of belief do not require extremal changes. That is, belief may increase to a value strictly less that 1, or decrease to a value strictly greater than 0.

Example 5.3. Let process q have a single internal event, s_q, and a single send event, S_a (corresponding to an "abort" message). Similarly, process p has an internal even, s_p, and a receive event, R_a. Process q's computations are either of the form s_q^* or one of: S_a, $s_q S_a$, or $s_q s_q S_a$. Conversely, p's computations are either of the form s_p^* or $s_p^* R_a$.

Now let b be the predicate "there are at least 3 events on q". This predicate is local to q. Let x be the computation $s_q S_a$. In this case, p **bel**$_1$ b **at** x. Extending this computation with a receive, however, decreases p's belief to a non-zero value: p **bel**$_{\frac{1}{3}}$ b **at** $(x; R_a)$.

The last example shows how internal actions may change a belief about a remote predicate.

Example 5.4. Let process computations for q and p be prefixes of $s_q s_q$ and $s_p s_p$ (internal skip events) respectively. Let $x = \epsilon$ be the empty computation and b be the predicate "there are at least 2 events on q". Then p **bel**$_{\frac{1}{3}}$ b **at** x. Extending x by an internal action s_p, we find that p's belief in b increases: p **bel**$_{\frac{1}{2}}$ **at** $(x; s_p)$.

The examples in this section illustrate how a process's belief in a remote predicate can increase or decrease as a result of sends, receives, or internal actions.

6. Applications of Belief

In this section, we consider two common constructs in distributed systems: leases and directories. Both constructs have proven to be very useful in the construction of real, asynchronous distributed systems, but a knowledge-based analysis does not accurately capture their utility. Instead, belief can be used to characterize these constructs.

6.1. Asynchronous Leases

A lease is a time-driven mechanism for releasing remote resources. It has been used to address a variety of problems in distributed system design, including cache consistency [8] and garbage collection [4]. Leases can be used to isolate a system from the failure of an individual component.

For example, in garbage collection, a reference is released only with an accompanying lease. The server guarantees the availability of the referenced object only for the duration of the lease. If the client wishes to maintain the validity of the reference it holds, the corresponding lease must be renewed with the server. If the client crashes, its lease eventually expires, and the server can safely garbage collect the referenced object.

Leases can be characterized using knowledge from the point of view of both the holder and the grantor of the lease. In the case of garbage collection, for example, granting a lease reflects knowledge that a remote reference exists. The expiration of a lease reflects the knowledge that a (valid) remote reference no longer exists.

In a purely asynchronous system, timeouts cannot be used reliably. In this case, the lease mechanism still has application, but the expiration of a lease triggers a message and acknowledgment protocol between the grantor and the holder. It is the reception of an acknowledgment from the holder that allows the grantor to know that the resource represented by the lease is no longer held by the remote client.

When failures can occur, however, the grantor cannot rely on an acknowledgment to learn something about the lease holder (since the lease holder may have crashed). In these cases, the grantor must be able to learn something from the *sending* of a message alone. This suggests that asynchronous leases should be modeled as reflections of *belief* rather than knowledge.

For a server p granting a lease to a client q, the property obtained is p **bel**$_\alpha$ q holds r (where r is a reference to the corresponding resource). The belief threshold α may be a function of the likelihood of q crashing. This belief may decrease by the local actions in p, modelling the decline of the corresponding belief as a function of time. The expiration of an asynchronous lease is also a local action at the server. The server's belief about a client holding a lease may also change (more dramatically) by a send action. After this send, a new belief property of the form p **bel**$_\beta$ $\sim$ (q holds r) is obtained. It is this belief that justifies garbage collection by the server.

Because belief (even with a threshold of 1) is not necessarily knowledge, holding (or granting) a lease is not a guarantee of availability (or unavailability) of the corresponding resource. This belief-based formulation reflects the best-effort semantics of real distributed leases, such as those provided in Jini [4].

Consider a basic asynchronous lease in a system with a server, p, (the grantor of the lease) and a client, q, (the holder of the lease). Events on q include: sending a request (S_{req}), receiving the granting of the lease (R_{grt}), receiving the expiration of the lease (R_{exp}), and sending an acknowledgement of the expiration (S_{ack}). The corresponding sends and receives are events on p. In addition, internal

events u_q and l_q represent client computation while holding the lease and while not holding it ("using" and "local" respectively). The protocol is illustrated in Figure 1.

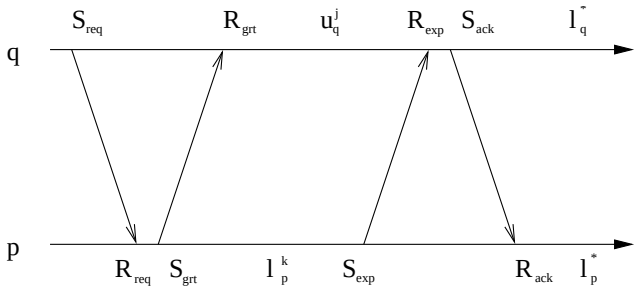

Figure 1. Messages Exchanged for an Asynchronous Lease

As seen in this figure, the client's computations are prefixes of

$$S_{req}R_{grt}(\epsilon|u_q)^j R_{exp}S_{ack}l_q^*$$

while the server's computations are prefixes of

$$R_{req}S_{grt}(\epsilon|l_p)^k S_{exp}R_{ack}l_p^*$$

To model the potential for client crashes, we add another client event, a_q (for abort). An abort can occur while holding the lease or after receiving an expiration notice but before sending the corresponding acknowledgement. The client computations are therefore prefixes of

$$S_{req}R_{grt}(\epsilon|u_q)^j(a_q|R_{exp}a_q|R_{exp}S_{ack}l_q^*)$$

Let b be the predicate "q is using the resource". Then b **at** x is true if x contains R_{grt} but does not contain R_{exp} or a_q. Let x be the computation $S_{req}R_{req}S_{grt}$. It is not the case that p **knows** b **at** x. That is, even after granting the lease, p does not know whether q is using the resource. However, p **bel**$_\alpha$ b **at** x with $\alpha = \frac{j+1}{2j+3}$. Notice that the value of α, which is close to $\frac{1}{2}$ for large j, is a reflection of the likelihood of crashes at the client.

Let x^k be the computation $(x; l_p^k)$ (the computation x extended by k local events on p). Then we find that p **bel**$_\alpha$ b **at** x^k with $\alpha = \frac{A}{2A+\binom{k+j+2}{j+2}+1}$ where $A = \binom{j+k+2}{j+1} - 1$. This quantity is asymptotically equivalent to $\frac{1}{2+\frac{k}{j}}$. Therefore, this belief goes to 0 as k goes to infinity, for a fixed j (see Figure 2).

We also compute that p **bel**$_0$ b **at** y for $y = (x; S_{exp})$, equivalently, p**bel**$_1$ $\sim b$ **at** y. Thus, after sending the expiration notice, p *believes* with 1 that q is no longer using the resource. In practice, p can determine when to send this notice according to its decay in belief that q is still using the resource. That is, when p's belief, as illustrated in Figure 2, decreases beneath some threshold, the lease expires. Thus, k functions as a timer, triggering the expiration of the lease.

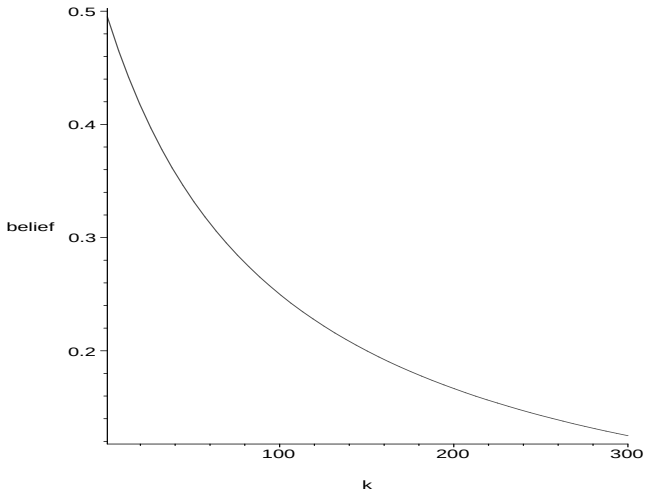

Figure 2. Decline of server's belief as a function of its local events (for j=50)

6.2. Distributed Directories

For bootstrapping a distributed application, components must find their remote counterparts. Typically, some form of naming service or directory is used (e.g. the CORBA naming service [16], the Java RMI registry [18], or UDDI for web services [14]). This model is based on the assumption that such a directory is more easily found than specific application components.

An entry in a directory is often modelled as a reflection what the directory *knows* about the remote application component. For example, an entry might include what interface is used to access the component and where the component is located. This knowledge is gained when the directory receives a message (a registration) from this component.

For real, loosely coupled, distributed systems, however, this knowledge-based model of a directory service is too stringent. The knowledge-based characterization requires that the directory can only "forget" information by sending a message. Thus, an application component that is registered with a directory service must maintain the same interfaces and be at the same location as indicated in its registry, until it receives a message from the directory! Thus, any information registered with a directory service (modeled by knowledge) represents a loss of volition on the part of the application component. This loss of volition is not appropriate for loosely coupled systems (*e.g.*, with high mobility, or independent reversioning of interfaces).

Even in loosely coupled systems, however, directories are still useful constructs. They are useful not because their entries represent knowledge, but rather because they represent *belief*. The remote application component may or may not have the attributes encoded in its registration, but the di-

rectory believes (with a certain threshold) that the entry is correct. Other components, by querying the directory service, would learn about this belief. For a component C using a directory D that has an entry r regarding a registered component, the property obtained is C **knows** D **bel**$_\alpha$ r. With this model of directories, application components can still change their state, so long as this possibility is reflected in the threshold (α) of belief.

7. Conclusion

In this paper we have proposed a formalism for the notion of belief in distributed systems based on isomorphisms of system computations. We have shown that this formalism is a generalization of the previously introduced and formalized notion of knowledge, which has proven to be useful in reasoning about distributed systems. Knowledge alone, however, is not adequate in loosely coupled distributed applications, where the required loss of volition is too strict of a requirement. We have demonstrated how the more general notion of belief can be applied to describe applications where knowledge is too restrictive. It seems likely that the notion of belief introduced in this paper can also be applied to some of the classical consensus problems in distributed systems, such as the coordinated attack problem [7], where a knowledge-based solution is known to be impossible.

8. Acknowledgements

The authors gratefully acknowledge the Distributed Components research group at The Ohio State University, as well as the anonymous referees, for numerous insights and helpful comments on earlier drafts of this paper. This work was supported by NSF ITR grant CCR-0081596, and an Ameritech Faculty Fellowship.

References

[1] Aumann, R.J. *Agreeing to disagree*, Annals of Statistics, 4:6, 1976.

[2] Chandy, K. M. and Misra, J. *How processes learn* Distributed Computing 1, 1986, pp. 40-52.

[3] Cresswell, M. J. *Logics and Languages*. London: Methuen and Co., 1973.

[4] Edwards, W. K. *Core Jini*. P T R Prentice-Hall, Englewood Cliffs, NJ 07632, USA, 2001.

[5] Fagin, R., Halpern, J. Y., Moses Y. and Vardi, M. Y. *Knowledge-based programs* Distributed Computing, 10:4, 1997, pp. 199-225.

[6] Friedman, N. and Halpern, J. Y. *Modeling belief in dynamic systems part II: revision and update* Journal of AI Research, 10, 1999, pp. 117-167.

[7] Gray, J. N. Notes on data base operating systems. In *Operating Systems, An Advanced Course*, LNCS, vol 60, Springer-Verlag, 1978, pp. 393-481.

[8] Gray, C. G. and Cheriton, D. R. Leases: An efficient fault-tolerant mechanism for distributed file cache consistency. In *Proceedings of the Twelfth ACM Symposium on Operating Systems Principles*, pp. 202–210, 1989.

[9] Halpern, J. Y. *Reasoning about knowledge: a survey*, Handbook of Logic in Artificial Intelligence and Logic Programming, Vol 4, D. Gabbay, C. J. Hogger, and J. A. Robinson, Eds, Oxford University Press, 1995, pp. 1-34.

[10] Halpern, Y. J. and Fagin, R. *A formal model of knowledge, action and communication in distributed systems*, Proceedings of the 4th ACM Symposium on Principles of Distributed Computing, 1985, pp. 224-236.

[11] Halpern, J. Y. and Moses, Y. *Knowledge and common knowledge in a distributed environment*, Journal of the ACM 37:3, 1990, pp. 549-587.

[12] Hintikka, J. *Knowledge and belief*, Cornell University Press, 1962.

[13] Lynch N. and Fischer M. *A lower bound for the time to assure interactive consistency*, Information Proc Letters 14, 4, June 1982.

[14] McKee, B., Ehnebuske, D., and Rogers, D. UDDI version 2.0 API specification. available at http://www.uddi.org/specification.html, June 2001.

[15] Moses, Y. and Shoham, Y. *Belief as defensible knowledge*, Artificial Intelligence, 64 (2), 1993, pp. 299-322.

[16] Object Management Group. *The Common Object Request Broker: Architecture and Specification*, February 2001. Revision 2.4.2.

[17] Parikh, R. and Krasucki, P. *Communication, consensus, and knowledge* Journal of Economic Theory, 52 (1), 1990, pp. 178-189.

[18] Sun Microsystems, Inc., 2550 Garcia Avenue, Mountain View, California 94043-1100. *Java Remote Method Invocation Specification*, revision 1.7, Java 2 SDK, v1.3.0 edition, December 1999.