Abort, Retry, Litigate: Dependable Systems and Contract Law

Hany E. Ramadan
University of Texas at Austin

The law will play an increasingly important role in the design of dependable systems, and designers of such systems will need to familiarize themselves with fundamental legal concepts. This note focus on Contract Law in the context of distributed systems.[1]

Nodes in distributed systems are constantly making promises to one another. While promises are generally kept, they may also be broken, either intentionally (e.g. for malicious or economic reasons), or unintentionally (e.g. hardware or software failure). An example is one node promising to store a file for another node, and to provide future access to it when requested. Traditional distributed systems were deployed within a single administrative domain (e.g. a bank) and thus the motivations to break promises were limited. The emergence of the Internet brought multi-party systems, usually organized as a server servicing multiple clients. Many of these services, such as IRC servers and web-mail sites, were provided at no cost. These free services were provided on a *best effort* basis, and broken promises were not a serious design constraint on such systems; clients simply retried.

A recent trend is Peer-to-Peer and Multiple Administrative Domain (MAD) systems, such as cooperative-backup and file sharing networks. Nodes in such systems are no longer simply clients, but also play a role in the provision of the service. The idea is that every node plays its part in serving the other nodes, in return for the benefits it receives from the service as a whole. There is often no centralized authority that owns or controls the system. One of the problems that can occur in such systems is that a node can become a *free rider*, where it enjoys the benefits of the service while refusing to contribute its own resources to the service.

This problem has been the focus of recent research. For example, the BAR model[2] treats nodes as rational actors, and provides them with incentives to behave properly. The assumption is that "without appropriate punishment, nodes may find it in their interest to misbehave". Thus, rational nodes are forced to keep their promises to avoid the severe sanctions that would befall them.

We approach the same problem from a different angle. We observe that the environment is not anarchic, and that enforcement mechanisms exist for certain promises. This approach is not incompatible with other approaches (e.g. BAR), but simply takes into account the legal context of the node's environment. The enforcement mechanism of interest is Contract Law, the main benefit and goal of which is the facilitation of transactions between entities which might have no other reason to trust each other. It is this benefit that we seek to take advantage of in dependable systems design.

A Contract is "a promise or a set of promises for the breach of which the law gives a remedy, or the performance of which the law in some way recognizes as a duty". In the U.S.[3] the ability to enter binding contracts is no longer limited to humans. The Uniform Electronic Transactions Act (UETA), enacted recently in most states, gives legal recognition to electronic records, signatures and contracts. It also declares that "A contract may be formed by the interaction of electronic agents of the parties, even if no individual was aware of or reviewed the electronic agents' actions or the resulting terms and agreements" (§14-b). This note reviews contract law and then considers the following questions for dependable systems: how are contracts entered, how are they breached, and what remedies are available? Of course, each system is unique and requires its own analysis to evaluate the contractual obligations arising in it.

A contract requires a bargain in which a promise is exchanged for a consideration. The consideration is either a return promise, or a performance [4]. The contract is formed when there is a manifestation of mutual assent to the exchange, which "requires that each party either make a promise or begin or render a performance".

One concern is: how exactly are contracts formed in a distributed system? Should they be between pairs of nodes, or multi-party contracts? Are two-phase commit protocols suitable for creating contracts? One principle that guides the law around offers and counter-offers is that periods where one party can speculate at the expense of the other should be minimized. Does this principle affect choice of agreement protocol, for example are coordinator-less protocols (e.g. Paxos-Commit) more suitable? The validity of a contract can be challenged on various grounds (e.g. misrepresentation, undue influence, duress, or public policy). Do these also occur in electronic contracts? For example, since nodes (unlike humans) may be able to analyze each other's code and foresee the consequences of *not* entering a contract, could this lead to duress claims in some circumstances?

[1] See Farnsworth. *Contracts* (Aspen, 2003), Radin et al. *Internet Commerce* (West, 2006), and Restatement of Contracts (ALI, 1981).

[2] Aiyer et al. *BAR Fault Tolerance* (SOSP, 2005).

[3] International contracts are outside the scope of this note.

[4] A performance is "a) an act other than a promise, or b) a forbearance, or c) the creation, modification or destruction of a legal relation".

Another concern is: what are the terms that define the scope of the contract? Will most contracts in the system be identical, with perhaps only a small number of "types" of contracts? Or will more numerous, fine-grained, customized contracts arise? Will contracts be standardized (fill-in-the-blanks), or will there be a process of negotiation? Humans are adept at creating new *options* in negotiations, what will computers do? Will there be a role for statistical machine learning, where nodes learn from past experiences? The most basic element of any contract are what the promise and consideration are. In a simple two-node cooperative backup system, perhaps the consideration is a return promise to store a similar amount of data, although more elaborate designs are possible.

How well specified will the contracts be? Usually contracts only account for the major contingencies, with the parties dealing with other issues as they arise. This is likely to remain the case for electronic contracts, as not only would the contingencies be too many to enumerate, but software has to be written to interpret and execute the terms, making it all the more unrealistic. Regarding contingencies, not every unexpected behavior is a breach. For example, if a party's private key is stolen, it may appear to breach its contractual obligations (especially if the thief is malicious). But if the theft is due to no fault of its own, this could provide grounds for defense to breach accusations. A system should be able to handle unanticipated contingencies, either by allowing humans to be "looped in" if and when such problems arise, or through some automated resolution mechanism. Immediately imposing sanctions on any breaching node can lead to increased liability, and may lead to a system that is not desirable to join in the first place (if these sanctions are irreversible), due to the many unforeseen circumstances that can occur.

A challenge in electronic contracts will be detecting, classifying and proving breaches. Breaches may be material or immaterial, and total or partial. We stress that freedom to contract goes hand in hand with freedom to breach contracts, and some so-called "efficient breaches" are even economically desirable[5], so systems should design for them accordingly.

One party may be at a significant advantage to establish the (non-)occurrence of a breach event. Perhaps the system should be designed such that such evidence must be made available to the other party to adjust the burden of proof. Trusted third parties that act as witnesses, may be helpful in system design. Such an approach was used in BAR, where a virtual witness was created out of the state machine. Perhaps real witness nodes (in the spirit of public notaries; or eNotarization) outside of the system can also be used. These can help mitigate certain forms of repudiation, help provide evidence for claims, as well as play a role during contract formation.

Cryptographic means to prove claims are also being put forward. BAR's Proof of Misbehavior can be generated by the "virtual witness", or may be generated by the breaching node (a signed confession). Other creative techniques to prove occurrence of events need to be investigated. To prove that a node possessed a file during a specific time period, perhaps it must produce portions of that file specified by a random, or an unpredictable data source (e.g. a stock market index). However, such an approach would not work if the file is publicly available at other sites, such that the breaching party could compute this data after the fact in response to a claim.

A primary principle of Contract Law is that remedies are aimed "not at compulsion of promisors to prevent breach, but for relief of promisees to redress breach". The relief usually comes in the form of payment of damages (cash), and occasionally in the form of specific performance (compelling performance). Penalties and other punitive damages are generally prohibited.

Damages are valued based on the cost to put the injured party in the position he would have been had the contract not been breached, the position he was in before entering the contract, or at a minimum to avoid unjust enrichment of the breaching party. We note that remedy valuation places constraints on how a system can respond to a breach: in general the remedy should be in proportion to the breach. In violation of this principle, a system which imposes severe sanctions in response to a minor breach, may be liable to the original breaching node, for the extra damage inflected upon it.[6]

What types of remedies will dependable systems use in practice: damages, specific performance, or both? Is dollar form the only acceptable form of damages or are there other forms of value [7]. Typically remedies are determined by litigation, but to avoid costly litigation most parties settle on their own, or rely on other processes such as Alternative Dispute Resolution. Automated resolution techniques would clearly be desirable for electronic contracts. Parties are guided during settlement by their estimate of the strength of their claims. How will nodes communicate with each other the strength of their claims, or the evidence they posses? When humans need to be involved, nodes should be able to provide the information needed to effectively handle the situation. Finally, class-action-like mechanisms may be needed, to pool together multiple small claims from many nodes against a breaching party, to provide a credible threat of litigation.

[5]Such as when the value of exploiting a sudden new opportunity outweighs the cost of compensating the injured party.

[6]For example, the BAR-B backup system scheme of severe sanctions, could result in all of a node's owned data being deleted as a result of a minor error by that node, e.g. returning even a single wrong byte.

[7]For example, in a file-sharing system, would a remedy perhaps be a transfer of file quota from the breaching to the injured party?