Promiserver: Procedurally Executed, Socially Enforced Microcontracts

by

Brent Paul Fitzgerald

Bachelor of Science in Symbolic Systems
Stanford University, 2001

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning,
in partial fulfillment of the requirements for the degree of

Master of Science in Media Arts and Sciences

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2007

© Massachusetts Institute of Technology 2007. All rights reserved.

Author

Program in Media Arts and Sciences
May 11, 2007

Certified by

John Maeda
Associate Professor of Design and Computation
E. Rudge and Nancy Allen Professor of Media Arts and Sciences
Thesis Supervisor

Accepted by

Dr. Andrew B. Lippman
Chair, Departmental Committee on Graduate Studies
Program in Media Arts and Sciences
Promiserver: Procedurally Executed, Socially Enforced Microcontracts

by

Brent Paul Fitzgerald

Submitted to the Program in Media Arts and Sciences, School of Architecture and Planning, on May 11, 2007, in partial fulfillment of the requirements for the degree of Master of Science in Media Arts and Sciences

Abstract

This thesis envisions the future of trust and social commitment in a highly connected society. Starting with a distributed, democratized labor force and economies of efficient niche production and consumption, we predict radical shifts in the meaning and methods of commitment and the institutions of trust. The central experiment of this thesis is Promiserver, a web-based service and toolset for creation of lightweight contracts—dubbed promises—that are written as code. The service decouples commitment logic from specific applications, providing a generalized tool and forum for dynamic creation, binding, and evaluation of promises. The goal of Promiserver is to facilitate new models of collaboration by offering a sensible, lightweight, and agile promise system as an alternative to traditionally heavy legal commitments.

Thesis Supervisor: John Maeda
Associate Professor of Design and Computation
E. Rudge and Nancy Allen Professor of Media Arts and Sciences
Promiserver: Procedurally Executed, Socially Enforced Microcontracts

Brent Paul Fitzgerald

The following people served as readers for this thesis:

Thesis Reader: Hiroshi Ishii
Associate Professor of Media Arts and Sciences
Program in Media Arts and Sciences

Thesis Reader: Brent C.J. Britton
Attorney in Intellectual Property and Corporate Law
J.D. Boston University School of Law
S.M. Media Arts and Sciences, MIT Media Lab
Acknowledgements

Many thanks to my thesis readers, Hiroshi Ishii and Brent Britton, for inspiring conversations and insightful, constructive feedback.

Thanks to all my friends and collaborators here at the lab, in particular the Physical Language Workshop crew for being great teammates these past two years: Burak Arikan, Mariana Baca, Isha Banga, Luis Blackaller, Kyle Buza, Yanglim Choi, Annie Ding, Amber Frid-Jimenez, Kate Hollenbach, Sanghoon Lee, Kelly Norton, Takashi Okamoto, Jun Sato, Danny Shen, and the whole JV team too. Extra special thanks to coaches Henry Holtzman, Amna Carreiro, and of course my intrepid advisor, John Maeda.

And finally, ultra special thanks to my family for your continued love and support, and a big mahalo to Jennifer Lucien for being my great friend and life travel companion.
A.2 Notable Promises ......................................................... 97
  A.2.1 finish thesis .......................................................... 97
  A.2.2 Paik’s Composition for Poor Man ............................. 97
  A.2.3 a little random promise ........................................... 98
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-1</td>
<td>Levels of Promises</td>
<td>21</td>
</tr>
<tr>
<td>2-2</td>
<td>Levels of Commitment</td>
<td>24</td>
</tr>
<tr>
<td>2-3</td>
<td>Second Life Permanent Record</td>
<td>31</td>
</tr>
<tr>
<td>2-4</td>
<td>The Long Tail</td>
<td>35</td>
</tr>
<tr>
<td>3-1</td>
<td>OPENSTUDIO Social Tagging &amp; Critique</td>
<td>49</td>
</tr>
<tr>
<td>3-2</td>
<td>OPENSTUDIO Transparent Processes</td>
<td>50</td>
</tr>
<tr>
<td>3-3</td>
<td>Original Hire-a-Designer</td>
<td>51</td>
</tr>
<tr>
<td>3-4</td>
<td>Hire-a-Designer</td>
<td>52</td>
</tr>
<tr>
<td>3-5</td>
<td>Clickback</td>
<td>53</td>
</tr>
<tr>
<td>3-6</td>
<td>Prototype v1: Natural Language</td>
<td>54</td>
</tr>
<tr>
<td>3-7</td>
<td>Prototype v2: Modular, Coded Clauses</td>
<td>56</td>
</tr>
<tr>
<td>3-8</td>
<td>Promise Lifecycle</td>
<td>58</td>
</tr>
<tr>
<td>3-9</td>
<td>Infrastructure</td>
<td>62</td>
</tr>
<tr>
<td>3-10</td>
<td>Profile Page</td>
<td>64</td>
</tr>
<tr>
<td>3-11</td>
<td>User Stats Graph</td>
<td>65</td>
</tr>
<tr>
<td>3-12</td>
<td>Participation</td>
<td>66</td>
</tr>
<tr>
<td>3-13</td>
<td>Live Parsing &amp; Evaluation</td>
<td>67</td>
</tr>
<tr>
<td>3-14</td>
<td>Security Example</td>
<td>69</td>
</tr>
<tr>
<td>4-1</td>
<td>Promise Tabs Interface</td>
<td>73</td>
</tr>
<tr>
<td>4-2</td>
<td>Daunting Blank Promise</td>
<td>74</td>
</tr>
<tr>
<td>4-3</td>
<td>Getting Help</td>
<td>76</td>
</tr>
<tr>
<td>4-4</td>
<td>State Variable Confusion</td>
<td>77</td>
</tr>
<tr>
<td>4-5</td>
<td>Personalized Promises</td>
<td>81</td>
</tr>
<tr>
<td>5-1</td>
<td>Promise Lifecycle with Negotiation</td>
<td>91</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Society is a web of commitments. We commit to provide for our children, parents, and loved ones. We commit to follow the norms and moral standards of our communities. We commit to pay taxes and obey the laws of the government. We commit to friends and teams by offering love, support, and effort. We commit to the organizations in which we work by showing up and doing our jobs. We commit to our market by using its currency, and investing in the constituting companies and funds. We commit ourselves everywhere, constantly, through our actions and our words.

And they commit back to us. This is how it works, how groups live harmoniously and get things done, through these mutual commitments. We promise one thing, we get back something else in exchange. In some cases the return is intangible, like a feeling of satisfaction or a sense of belonging. Often it is very tangible, like a paycheck or a home.

The commitments and the returns are not necessarily straightforward, often involving many people connected via chained, cascading promises [21]. These commitments depend on one another, cumulatively forming a networked structure within which more commitments are made. We make these commitments when we think we stand to gain from them, and the web of commitments—society—has evolved rules and institutions to self-reinforce this behavior. From simple needs and relationships emerge complex, self-structuring networks of promises.
These networks exist in all societies and governments, though the connections and topologies may differ radically. In an oppressive regime, we may commit to actions that we would not inherently want to do, but that we believe we cannot afford not to do, and our commitments may not lie where we would like. In a free, dynamic society, an ideal society, we may commit to actions that we believe will return something that would be costlier or impossible without the commitment. Often these commitments are implicit. Sometimes they are spoken or written promises. And increasingly, as there is more to lose and more to gain, they are legally binding contracts.

This thesis project explores the collision of this fundamental glue of our social existence with the radical paradigm shifts engendered by connecting technologies. What happens when we commit ourselves to people, groups, corporations and ideas that we will never meet, and that operate on a layer beyond our community norms or our national economies and politics? What happens when we wish to collaborate, to pioneer new ideas and movements, and to mutually capitalize on one another’s strengths, outside the boundaries of real space and real time, and beyond the scope of any single nation’s law? How do we know who to trust? How do we prove ourselves trustworthy? What is the language of trust?

1.1 Starting Points

Issues of trust and commitment are as old and deep as any. These are questions about humanity and civilization, and we find answers only by looking at our own practices, by studying our own institutions of trust. Yet the goal of this thesis, and of the Media Lab as a whole, is not only to thoroughly understand the present, but to use this understanding to invent the future. This thesis imagines our future practices in a networked society, with a distributed, democratized labor force, and economies and communities of highly efficient niche production and consumption. From this image we begin to see radical shifts in the meaning of commitment and the institutions of trust.
In nearly two years at the MIT Media Lab, I have worked in a team—the Physical Language Workshop\(^1\)—dedicated to innovating and exploring these visions of the future. It is through this work that I have begun to find common factors of trust and commitment.

During this period, the primary mission of the PLW has been invention of new tools, methodologies and models of creative collaboration. As a team, we began with a vision of a marketplace for digital creators, an online space in which a community develops not only around the process of making, but the equally core acts of buying, selling, trading, and investing in digital content and services.

From this ongoing project, the group research agenda bloomed into a variety of offshoots: ad-hoc composition of networked electronics, human creative computation, organic and participatory advertising, community coding systems, anonymous content publishing, fake online identities, authentication alternatives, and low-barrier phone interfaces. Each project further crystalized a slice of our vision of a networked future of collaboration.

### 1.2 Promiserver

The central experiment of this thesis, and the engineering and design side of this work, is Promiserver, an online application and service for a unique form of social, lightweight contracts. Promiserver is a web based application in which people may write commitments in code rather than natural language. These commitments are referred to as promises, and once published and signed they are continuously evaluated until complete with a breach or success, or unanimously cancelled by participants. The outcomes of promises are publicly associated with their participants, viewable and searchable in their online, permanent records.

Promiserver is an alternative to the legal tradition, offering a tool and forum for socially binding promises rather than legally binding contracts, and decoupling commitment logic from specific applications. Though promises

\(^1\) Physical Language Workshop (PLW) - http://plw.media.mit.edu/
are code, they offer as much rigidity or flexibility as the authors intend, and are not constrained by the system itself. All promise code is open source, subject to public critique, and freely available for reuse.

Designed as a platform for evaluation of commitment logic, a social commitment system like Promiserver can be incorporated into a wide range of transaction or contract models, from auctions to e-lancing to social obligations or political pledges. At base it is a public forum for people to make declarations about their intended actions, and to commit themselves by relinquishing control not to an expensive, slow legal system, but to evaluation of open code and public critique by peers.

Promiserver is on the web and open to the public. As of this writing it can be found online at http://promise.media.mit.edu/

1.3 Thesis Structure

This thesis is divided into five chapters, this introduction being Chapter 1. Chapter 2 then develops terminology and surveys related theory, developments and projects from philosophy, linguistics, computer science, politics, and business. Chapter 3 starts with an overview of earlier experiments that motivated Promiserver, then details the design criteria and engineering of the system. In Chapter 4 we report on usage of the system, evaluating strengths and weaknesses of the implementation, then looking more broadly at the concept as a whole within the scope of topics presented in Chapter 2. Finally, Chapter 5 is a summary of lessons learned, and a look towards the future.
Chapter 2

Background

A dynamic society... depends not only on preserving fluidity but on permitting permanence. To learn, we must experiment. But to experiment, we must commit ourselves. And we must find ways to cooperate with others, to extend trust.

Virginia Postrel, The Future and Its Enemies [27, page 130]

In this chapter we define promises and contracts, and people’s motivations to participate in them. We will look at the use of promises and commitments in the law, commerce, communities, and collaborative systems. All these areas are fundamentally linked, since all build and rely on one another. Along the way we will define terms and frameworks that will later be useful in discussing the conceptualization, design and analysis of Promiserver.

2.1 Promises and Contracts

We encounter and participate in promises throughout our lives. It would seem that we should know exactly what it means to make a promise, and that we would understand why we make them. Yet, like most apparently simple topics, the concept of the promise actually proves to be very complex, unfolding and intersecting into disciplines ranging from law to logic to economics.
Going by the old analogy of building a house, the topic of promises is a deep, dark and old woods where we need to get our supplies, but that we may get lost in. In this section we won’t go far, just a little trip in to gather what we need so we can lay a solid foundation for the rest of the thesis. We will nail down some definitions of promises and contracts, analyze what they give us, both individually and collectively, and look at some related theories that will later come in handy for framing the target applications of this thesis.

2.1.1 Definitions

What is a promise, and what is the difference between a promise and a contract? Our standard definition of a promise is an assurance someone makes about her future actions. The Oxford English Dictionary definition is as good as any:

A declaration or assurance made to another person with respect to the future, stating that one will do, or refrain from, some specified act, or that one will give or bestow some specified thing. (Usually in good sense, implying something to the advantage or pleasure of the person concerned.) [28]

This definition is obviously quite broad. As the parenthetical portion implies, promises are generally made in a cooperative, beneficial spirit, though this is not a required element of the definition. We generally specify a promise as a contract when it is explicitly legally binding, meaning that the promise is enforced by the legal system. A dispute or breach of contract may result in legal recourse. Atiyah’s classic introductory text provides a concise definition:

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 recognises as a duty. [3, page 23]

Typically we stick to the term promise for a social or informal assurance, like promising to do the dishes, keep a friend’s secret, or perform a chore. The caveat here is that many promises—even those we don’t label as contracts—may be potentially legally binding. In other words, and in
accordance with Atiyah’s definition, contracts are promises, and promises may or may not be contracts. The set of legally binding promises is a subset of all promises, but is a superset of contracts, as shown in Figure 2-1.

![Figure 2-1: Levels of Promises. Not all promises are legally binding, and not all potentially legally binding promises are contracts, but all contracts involve promises and are potentially legally binding.](image)

Legally, a promise may be enforced as a binding contract only if there is an exchange. This exchange is **consideration**: a promise or offering made in return. Yet in actuality we participate in many promises with consideration but that we do not grant legal power, such as social commitments. These promises are still enforced, but via social pressures rather than a legal system[5]. Promiserver is an exploration of this type of enforcement, attempting to stretch it out of the purely social realm.

In contrast to legal contracts, Promiserver’s binding is simply based on participants signing to the code, regardless of the form of the code and the commitments implied within. Promise code does not specify who is committing what or to whom; it only specifies the logic of the conditions that must be met to arrive at certain outcomes. While the use of a programming language requires adherence to a certain syntax, Promiserver places no requirements on the actual meaning of the code. Promises simply
describe the logic of outcomes; delegation of responsibility is left to the participants and witnesses to decide.

2.1.2 The Social Contract

This thesis is driven in large part by philosophy and political theory, particularly the tension between top-down organized rule systems and bottom-up individual relationships. How do the individual bonds and commitments we make to one another form communities, societies, and governments?

The central tenet underlying Promiserver is that human relationships can be modeled as contracts, and that this contract approach is helpful in both understanding and constructing social systems. This treatment of human networks as systems of contracts is not a new idea, arguably going back at least as far as Plato’s Republic, which prescribes citizens’ roles, rights and responsibilities within the ideal nation state, and the benefits these citizens gain by their participation. Contracts took on far greater importance in the age of Enlightenment, in particular with social contract theory as developed by political philosophers Locke, Hobbes, and Rousseau. [8]

Social contract theorists use the contract as a model for both explanatory and normative theories of society and government. These theories are collectively classified under contractarianism[1], defining the contract as a pact in which a participant voluntarily gives up some freedom in exchange for some return. The social contract is the base, atomic agreement that has lifted people from the hypothetical state of nature—the natural world without law or government—to organized, rule-bound society. Specifics of these social contract theories differ, such as the Hobbesian model of the anarchic, immoral state of nature, to Lockean rational and moral version of the same. These variations in starting points lead to widely divergent conclusions among the various theories.

Yet fundamentally all these theories follow a similar empiricist approach of dividing the problem space into discrete states, and from there examining the logic at the micro-level—the rational individual—in order to understand the
behavior of the whole—society. The social contract is the unit-level, base relationship which participants form with others, and consequently with the society that emerges from others doing the same.

2.1.3 Speech Act Theory

Early social contract theory came out of a period in which empirical, natural philosophy approaches were first gaining widespread recognition and social validity. The early influential social contract theorists advanced their political agendas using empirical approaches. They started at the individual, and used the contract as a mechanism to generalize upward to normative theories of government.

Since then, science and analytic philosophy have splintered off from the normative, instead looking deeper into natural logic and structure. Theorists in computer science and linguistics have developed analytic theories of the underlying mechanics of commitment. **Speech act theory**, originating with J. L. Austin and further developed by Searle, is one such theoretical tool that has proven useful in understanding the commitments implicit to all language. Speech act theory casts every conversational act as a commitment to a course of future events, divided among five classes [35]. Winograd concisely sums up Searle’s “five fundamental illocutionary acts” in [34], reprinted below:

- **Assertive**: Commit the speaker (in varying degrees) to somethings being the case—to the truth of the expressed proposition.
- **Directive**: Attempt (in varying degrees) to get the hearer to do something. These include both questions (which direct the hearer to make an assertive speech act in response) and commands (which direct the hearer to carry out some linguistic or non-linguistic act).
- **Commissive**: Commit the speaker (again in varying degrees) to some future course of action.
- **Declaration**: Bring about the correspondence between the propositional content of speech act and reality (e.g., pronouncing a couple married).
- **Expressive**: Express a psychological state about a state of affairs (e.g., apologizing and praising).

Of these five, collaborative work depends most heavily on directives and commissives, and in fact this is the approach of Winograd and Flores, who set up speech act theory as an opposing design philosophy to the rationalistic tradition permeating computer science and artificial intelligence. They use speech acts as the foundational element of their Coordinator system, an office communication and management system [13].

Though not an original influence in the design of Promiserver, the Coordinator's underlying philosophy is remarkably similar. Promiserver arguably follows in this tradition of language acts, but deviates in its emphasis on trust and community reputation in commitment. We will further discuss Coordinator in the section on collaboration systems, page 42.

Through the lens of speech act theory, our classes of promises get an outer layer, so they are now encompassed by all speech acts, as in Figure 2-2.

---

Figure 2-2: Levels of Commitment. Speech acts are the outermost class of commitments. The distinction between a promise speech act and a non-promise speech act is not always obvious.
2.1.4 Promise Institutions

Why do we make promises, and why do we keep them? At first glance promises seem like an innate component of human life, something we engage in every day. Yet as Hume so aptly puts it in *A Treatise of Human Nature*, “A promise wou’d not be intelligible before human conventions had establish’d it.” [6, p14][17] The practice of making and participating in promises is a result only of human interests and behavior. Promises are accepted and embraced as a cornerstone institution of modern society. So what do we gain from this institution? Why are they in our best interest?

The gain is simple: trust. Promises make possible collaborations that benefit participants more than if they worked alone. Economics and law theorist F.H. Buckley addresses this topic thoroughly:

What the institutions of promising and contract law crucially supply is the element of trust which makes promises credible and permits promisees to rely on promisors. Without the trust created by contract law, opportunities for gain from joint projects would be lost and our society would be poorer” [6, page 35].

Buckley’s explanation uses a classic model from game theory, the Prisoners’ Dilemma (Table 2.1).

<table>
<thead>
<tr>
<th>Player 1</th>
<th>Cooperate</th>
<th>Defect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Player 2 Cooperate</td>
<td>3,3</td>
<td>-1,4</td>
</tr>
<tr>
<td>Defect</td>
<td>4,-1</td>
<td>0,0</td>
</tr>
</tbody>
</table>

Table 2.1: The classic Prisoners’ Dilemma (no contract enforcement). Mutual cooperation yields the highest overall payoff, yet defection is the optimal individual strategy because there is no trust between the players. [6, page 36]

The archetypal Prisoners’ Dilemma problem puts two equal participants in a situation in which each may choose to work together (cooperate) or not (defect). Mutual cooperation results in the highest total payoff (3,3), and may at first seem to be the preferred strategy. Yet cooperating while the other player defects results in incurring a penalty while the other player benefits (-1,4), which is undesirable. If both defect then neither achieves any benefit.
(0,0). Assuming each participant is rationally self-interested (acts solely in her own best interest) and isolated (unable to communicate outside the game or with one another), defection is the optimal individual strategy.\(^1\) So, given two rational, independent, and self-interested individuals, in this model situation there is a natural aversion to cooperation.\([6, pages 36–38]\)

Consider the classic contract theorist state of nature, in which each person tries to maximize gain, but there is no trust among parties. There will be many cases like the Prisoners Dilemma in which cooperation is the optimal group strategy, but the individual strategy is uncooperativeness.

Promises change this equation. Or, more specifically, promise institutions change it. A promise institution is a context in which commitments are created, and that provides some enforcement of these commitments, in the form of remedy\([5]\) if they are broken. By deterring greedy behavior, cooperation becomes the preferred individual strategy. So a promise institution can change the prisoners’ dilemma outcome, aligning the individual strategy with the group strategy, as in Table 2.2.

<table>
<thead>
<tr>
<th>Player 1</th>
<th>Cooperate</th>
<th>Defect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cooperate</td>
<td>3,3</td>
<td>2,1</td>
</tr>
<tr>
<td>Defect</td>
<td>1,2</td>
<td>0,0</td>
</tr>
</tbody>
</table>

Table 2.2: The Prisoners’ Dilemma with contract enforced and damages awarded to the cooperator and punishment to the defector. Cooperation remains the highest overall payoff, and is also the preferred individual strategy. Individual and group interests are aligned.\([6, page 38]\)

The workings of the promise institution may vary considerably. I propose a division of the set of institutions into two core classes: social and legal. Each operates differently, yet they overlap considerably in their scope. We use a social promise institution when we make promises with friends or family,

\(^1\) There are many Prisoners’ Dilemma example situations. In some versions, mutual cooperation results in a positive outcome, while in others mutual cooperation simply avoids a negative outcome. The namesake Prisoners’ Dilemma scenario places suspected criminals separately in captivity, each facing interrogation and the choice of whether to defect—confessing and ratting out their partner—or cooperate with the partner by maintaining innocence. If both maintain innocence, they go free. If one confesses and the other does not, the defector is rewarded and given freedom, while the other is punished and put in captivity. If both confess, both are imprisoned. There is no communication between the prisoners. As a result, the ideal group strategy (the strategy with the overall highest payoff) would be to maintain their allegiance and claim innocence. Yet the preferred individual strategy (the strategy with the individual highest payoff regardless the other’s strategy) is defection. This divergence between the group strategy and the individual strategy is the Prisoners' Dilemma.
or within a community. These are agreements that we would never expect to go to court. The social promise institution’s enforcement mechanism is gossip, guilt, shaming, or decrease in community standing or reputation.

We invoke a **legal promise institution** in matters of government or business, where the social enforcement lacks sufficient scope or strength to ensure cooperation. This legal institution is **contract law**. It offers far greater trust firepower because remedy is executed via the government judicial system, and for this reason contract law is the institution used by corporations and governments when large transactions are at stake.

Yet in practice, participating in a legal contract often involves lawyers, fees and time for at least one party, and potentially severe legal ramifications for breach. So this high level of trust comes at a high cost: high barrier to entry, far greater stakes, and reliance on a creaky, slow, and often unpredictable judicial system. For some situations, legal contracts are perhaps too heavy, and social contracts too informal. Is there an intermediate?

### 2.2 Code & Law

What is the role of law and legal contracts as an institution in our connected, digital society? How does law relate to digital technology and culture, and do technology and law change interact? We must address these questions because Promiserver is, in essence, an alternative to the legal tradition, in language as well as the institution it attempts to build to provide trust.

Again, this is another deep area, and an expanding one. Not only is technology constantly developing, but so too are our technology practices, and the laws surrounding these practices. We’ll survey this evolving landscape, in particular the tensions of control, and the architecture of code usurping other forms of regulation.
2.2.1 Code is Law

We start with Lawrence Lessig’s *Code* [20], the landmark analysis of the architecture of computers and the internet as an implicit, often overlooked, and increasingly powerful regulating force. This architecture is code, and its design determines the nature of our experience. There are three other regulating forces as well: law, the market, and social norms (Table 2.3). All these regulators interact with and influence the development of one another.

<table>
<thead>
<tr>
<th></th>
<th>Regulation Mechanism</th>
<th>Temporality</th>
<th>Invocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Law</td>
<td>Threat of Legal Punishment</td>
<td>Reactive</td>
<td>Agent</td>
</tr>
<tr>
<td>Social Norms</td>
<td>Threat of Community Punishment</td>
<td>Reactive</td>
<td>Agent</td>
</tr>
<tr>
<td>Market</td>
<td>Price</td>
<td>Preventive</td>
<td>Agent</td>
</tr>
<tr>
<td>Architecture</td>
<td>Options</td>
<td>Preventive</td>
<td>Self-Executed</td>
</tr>
</tbody>
</table>

Table 2.3: Lessig’s four regulators: law, norms, market, and architecture. [20, pages 340-5]

Consider how architecture is different. Once created, architecture’s constraints remain in effect without any person overseeing or invoking them. In Lessig’s terminology, architecture is self-executed. Its constraints are also objectively present before any transgression occurs. It exists in the world, not merely as subjective constraints from fear of punishment, but as hard boundaries preventing certain actions.

Architects design forms and dimensions that regulate the experiences of people and groups within a space. Likewise, programmers decide the representations of data and the flow of interactions within software. Instead of creating physical constraints, they create code constraints. When this code hosts a social system, its form determines all aspects of people’s lives as they interact with and within the space.

If we are faced with creating or maintaining a social system, as designers we use one or more of these regulating forces to shape the experience of the participants. On the net, as Lessig points out, the most accessible and direct method of regulation is not through laws, the market or institution of norms, but through the architecture. In essence, all these regulatory forces get wrapped up into the code.
2.2.2 Second Life

Lessig provides many examples to demonstrate and expand on this point of the salience of architectural regulation in cyberspace. We’ll look at Second Life\(^2\), a popular 3D, online virtual world, within which people socialize, build, and exchange goods, services, and currency.

As with all social software systems, Second Life’s code is an instantiation of the choices made by its designers, regarding not only architecture (such as the physics of allowing people to fly), but also social, market, and legal regulation. These regulators are embedded and enforced within the code.

**Property Flyovers**

As an example, Lessig cites the option granted to property owners to disallow other people from flying over their land at a height less than fifteen meters. This code/law for flyover heights was not necessary at first. The first version of Second Life didn’t even have any code for property. That was added later, after which the creators found it necessary to make rules to protect that property.

In real space, the law means you can be penalized for violating the “high/low” rule. In Second Life, you simply can’t violate the 15-meter rule. The rule is part of the code. The code controls how you are in Second Life. There isn’t a choice about obeying the rule or not, any more than there’s a choice about obeying gravity.

So code is law here. That code/law enforces its control directly. But obviously, this code (like law) changes. The key is to recognize that this change in the code is (unlike the laws of nature) crafted to reflect choices and values of the coders.

This example is particularly interesting because its motivation is similar to that of the laws in the real world, but its implementation is in architecture

---

\(^2\) Linden Lab’s Second Life - http://secondlife.com/
instead of law. See Lessig’s more thorough analysis of the architecture of Second Life. [20, pages 106-11]

Data Trails

In conjunction with recent research initiatives at the MIT Media Lab, I recently performed my own investigations in Second Life, looking at residents’ ownership rights of the data they create through their everyday actions.

Within Second Life, and unlike reality, all actions are manipulations of information. The architecture makes possible explicit ownership, trade, and manipulation of certain types of information, like avatar clothing, property, or inventory objects. Other information is an implicit result of existence and action within the world, such as location coordinates, chat text, or appearance history. Most of the information in Second Life is of this latter type. You can’t help but generate it.

In the real world, gathering and aggregating this sort of implicit, passively generated information is often referred to as “spying” or “stalking.” Not only does this sort of behavior typically break the law and social norms, it is also extremely difficult in the real world architecture.

Second Life’s architecture makes it easy. I created a short piece of Linden Scripting Language (LSL) code that can be installed on any owned object (even invisible ones), and sends all overheard conversations to an external server, appropriately named The Permanent Record, where it is displayed in real time, as well as aggregated by speaker and location.

Second Life’s API makes it possible for a script to not only record conversations, but easily identify all speakers by unique ID and name, as well as location, time, and even other variables like velocity. In a divergence from the architecture of anonymity underlying the internet, the Second Life architecture is strongly focused on identity.

As more of these virtual worlds and social systems spring up, each will bring its own set of code-enforced values, and those that gain popularity will
continue to evolve their architectures. The most dynamic systems will be the ones that create the most open architectures, permitting the community to form its own norms, markets and even laws.

2.2.3 Dynamist Rules

These two stories are examples of the shift of architectural code overtaking regulations that were once the domain of law, social norms, and the market. These shifts are perhaps inevitable, and this thesis does not condemn the change, but we do aim to understand and capitalize on it in pursuing new models of collaboration.

Code has far different characteristics than other regulating forces, and as more of our lives move online, it becomes a dominant regulator. Dynamic, open communities and economies online depend on dynamic architecture. How then, do we construct code/architectures that facilitate fluidity, collaboration, and innovation?

The key may be in creating an underlying architecture that consists only of rules that encourage fluidity in the higher level regulators. Virginia Postrel lays out five dynamist rules [27, page 116]:
• Allow individuals (including groups of individuals) to act on their own knowledge.

• Apply to simple, generic units and allow them to combine in many different ways.

• Permit credible, understandable, enduring and enforceable commitments.

• Protect criticism, competition, and feedback.

• Establish a framework within which people can create nested, competing frameworks of more specific rules.

The aim of Promiserver is to provide a space in which people create their own private rules, with minimal outside regulation on those rules, yet a focus on commitment and critique within the system. Postrel’s dynamist approach offers a set of axioms for designing exactly these sorts of dynamic, self-regulating community systems.

2.3 Commerce

Promises have always been a prerequisite for commerce. Each exchange of money, goods, services, or information carries a risk that one partner in the transaction may renege or fail to deliver. Depending on the institution, promises provide some level of security for people and groups to depend on one another to fulfill obligations in the future. Traditionally companies have used legally binding contracts, invoking the judicial promise institution (page 26), instilling a higher level of trust by incentivizing cooperation. Legal contracts are a form of private law, an additional layer of rules that people and corporations may voluntarily create and enter into.

Technologies such as the web and mobile communications sparked a revolution in business. In particular we are witnessing a shift to smallness and agility, with a renewed vigor in small businesses and cottage industries. We are also seeing downsizing and outsourcing of work among larger companies. As the nature of business changes, we are starting to see
possibilities for whole new models of commerce, and new paradigms for market-based collaboration and commitment.

2.3.1 eLancing to Free Agents

As the web hit its first boom in the late nineties, and as telecommuting, desktop publishing, and mobile communication became more commonplace, researchers laid out predictions about what these practices held in store for the future of business. Two MIT faculty published a seminal paper *The Dawn of the E-Lance Economy* envisioning a radical future of dynamic, decentralized, and market-driven organization and management (see also [25, page 38]). Big business will be out, they argued, and small will be in:

When it is cheaper to conduct transactions internally, within the bounds of a corporation, organizations grow larger, but when it is cheaper to conduct them externally, with independent entities in the open market, organizations stay small or shrink.[22, page 147]

The recent revolution in digital technology allows for vastly more efficient communication and exchange. It becomes just as easy to work with an outside contractor as someone internal to the company. These changes shrink the benefit of keeping work in-house, and instead encouraging companies to outsource to more specialized firms, whose rates and practices are subject to the competitive market. As this transition occurs, they argue, new “rules of the game” will come into play:

One of the things that allow a free market to work is the establishment and acceptance of a set of standards—the “rules of the game”—that governs all the transactions. The rules of the game can take many forms, including contracts, systems of ownership, and procedures for dispute resolution. Similarly, for an e-lance economy to work, whole new classes of agreements, specifications, and common architectures will need to evolve.[22, page 151]

The rest of this section follows the development of these new rules as these ideas have begun to matriculate from the realm of hypothetical scenarios to real business plans. The aim of this thesis is not to implement some system
according to these rules, but to create a dynamic space in which they may be written by others.

Armies of Free Agents

Not long after its inception among academics, this eLance vision of technology’s impact on business began to find wider adoption. Technology pundits have been predicting and documenting on these technologically-driven shifts for some time. Daniel Pink’s 2001 *Free Agent Nation* [26] reported on the increasing number of people moving into self-employment, and the resulting changes in culture, business, and government. Pink asserts that this shift will result in a radically different landscape, not only economically, but politically and legally as well. Similarly, blogger/pundit Glenn Reynolds has further explored this collective, democratized theme, characterizing it as:

a dramatic reversal of recent history, toward more cottage industry, more small enterprises and ventures, and more empowerment for individuals willing to take advantage of the tools that become available. We’re likely to see a movement from the impersonal, imposed means to an end to a more individualized, grassroots way of doing things [31, page 9].

What the eLance proponents envisioned, and what Reynolds and Pink show evidence for, is that more and more people have the tools to run their own independent businesses. With its democratized approach to social contracts and its low barrier to entry, Promiserver—and the vision of micro-contracts in general—may be one of the game changing new rules.

2.3.2 Promises of the Long Tail

Chris Anderson’s 2006 book *The Long Tail* [2] resonated with technologists and business leaders with its hypothesis that the internet catalyzes new economies in which many niche products collectively form a much larger
market than the top popular products. This niche market is the long tail (Figure 2-4). Businesses taking advantage of the internet, he argues, can offer long tail products alongside mainstream products for minimal extra cost. These offerings result in profound cultural and economic shifts.

Anderson identifies three technological and sociocultural changes that power the transition to long tail economies [2, pages 54-57]:

1. **Democratizing the tools of production.** More stuff gets created by more people, resulting in more niche content and products, which lengthens the tail. Anderson refers to these people as producers.

2. **Cutting the cost of consumption by democratizing distribution.** The internet gives consumers better access to the niche content and products, making the tail fatter. People and businesses facilitating this distribution are aggregators.

3. **Connecting supply and demand.** Services help consumers find what they will like. These services are filters.
Promiserver also banks on the long tail, but a somewhat more general version of the hypothesis than Anderson’s. Despite all the discussion of niche culture and markets, Anderson focuses primarily on consumers and entertainment media, and his examples are often modern oligopolies like Apple iTunes, Amazon, Netflix, Google, and EBay. These are corporations that have managed to become one-stop shops for the niche.

Given the Daniel Pink vision of a growing class of free agents, let’s recast Anderson’s three forces of the long tail to look not only at niche media and products, but niche services as well. The basis of Promiserver is a belief that these technology forces—for production, aggregation, and filtering of the niche—will usher in revolutionary democratization of how people socialize and work, bestowing improved efficiency and mutual capitalization on one another’s specialties and strengths. Anderson writes of the coming revolution as a multifaceted fragmentation into microcultures:

> The same Long Tail forces and technologies that are leading to an explosion of variety and abundant choice in the content we consume are also tending to lead us into tribal eddies. When mass culture breaks apart, it doesn’t re-form into a different mass. Instead, it turns into millions of microcultures, which coexist and interact in a baffling array of ways. [2, page 183]

Though distributed and refactored, these long tail microculture interactions are still fundamentally familiar human relationships, like friendships, alliances, affinities, conspiracies, support groups, startups, partnerships, etc. The forces of the long tail are efficiency-generating production, and relationship enablers and accelerants. As the opportunities for interaction increase, so too does the need for new approaches to managing and navigating these relationships. As we’ll further discuss in Chapter 3, Promiserver is a first attempt at such a new approach.

### 2.3.3 Industries of Trust

In this section we’ll look at examples of these new, bottom-up business approaches, and how they each depend on or relate to commitments. When
we start to look at the new models of online commerce, we see non-legal commitments everywhere, generally hard-coded in.

Payments

We’ll start with PayPal³ and the online payment industry, which has in some respects served as a model for Promiserver. PayPal has taken the financial exchange implicit in all e-commerce, and factored it out into a separate, standalone service with an API for many other systems to use it. These services have vastly reduced the barrier to entry for online commerce. Individuals and small businesses can now easily receive and exchange payments without setting up any infrastructure or even writing a line of code. It is the financial backbone of eBay’s marketplace, as well as countless small companies doing business in the online economy.

PayPal removes the burden of payment infrastructure from individual participants in the market, effectively lowering the barrier of entry into the marketplace. This design is similar to Promiserver’s goal of generalizing and factoring out the contractual commitment and trust components in collaborative systems, and moving this functionality to a trusted, third party social promise institution. PayPal facilitates the use of their services by offering both a web interface and a rich web services API, so developers are able to integrate it into their own sites and services. This approach is also core to the Promiserver design.

The parallels end, however, when it comes to the handling of the core logic of the transaction. While PayPal and similar systems make it possible to send, request, or receive a payment, they lack the capability to capture more complex procedures. If we want to send or receive a payment only on a certain condition, such as passing of a certain date or completion of some external event, these current systems do not provide this functionality. The code describing the payment logic is inextricably coupled with the rest of the system code.

³ PayPal - https://www.paypal.com/
Peer-to-Peer Lending

Adapting the humanitarian principles of microfinance\(^4\) to a more democratic, technology-driven lending system, Kiva\(^5\) is a non-profit company that connects entrepreneurs in the developing world with lenders in first world. It is an online system, posting information about borrowers, and allowing lenders to choose which people to lend to. In making the loan, the two enter into a lightweight contract. As with many microfinance systems, the enforcement lies at the community level, as loans are administered by local non-profit foundations.

Meanwhile, companies such as Zopa\(^6\)[32] and Prosper.com\(^7\)[10][15] are for-profit microfinance, providing online marketplaces to connect borrowers with lenders. The approaches of these companies differ in the details, but neither offers insurance or safeguards for lenders. Instead, each provides a reputation system, indicating the trustworthiness of borrowers by credit reports similar to the ratings used by traditional lending institutions.

Interestingly, these peer-to-peer loans were actually part of Pink’s Free Agent Nation vision back in 2001. He called them F.A.N. bonds, and even hypothesized a credit rating system similar to the ones used by sites like Prosper. As evidence, he pointed to the transition of commercial lending from bank loans to corporate bonds to junk bonds, and predicted a parallel process happening in personal finance, from credit cards to individual, free agent bonds. [26, pages 275-280]

Auctions

As an early, highly visible and successful dot-com with consistent, sustainable revenue, EBay remains one of the most fascinating and active web-based social systems. Hard-wired into the code of EBay’s auction and marketplace systems are implicit, lightweight contracts specifying the rules and logic of the auctions and followup transactions.

---

\(^5\) Kiva P2P Humanitarian Microfinance Lending - http://kiva.org
\(^6\) Zopa P2P Lending - http://zopa.com/
\(^7\) Prosper P2P Lending - http://prosper.com/
For participants in these auctions, breach of contract is remedied within the community using a ranking system. After a transaction is complete or successful, both participants may rank and comment on one another’s performance, based on factors such as timeliness of payment or shipping. These rankings are cumulatively designed to reflect a user’s standing within the community, incentivizing participants to keep their promises. Research indicates that EBay’s reputation system is effective, with seller reputation having a statistically significant relationship to price fetched. [30] [16]

Like many other online social systems, Promiserver uses a community enforcement model that echoes EBay’s reputation system. Yet while Promiserver embraces this reputation approach, it also factors out the contract system, making explicit the contracts that are implicit to systems like EBay, and providing a means to aggregate reputation, trust and identity across multiple systems. In this sense it is a drastic departure.

A differently structured auction system is used by Google AdWords, which automates auctions to determine which advertisements should be placed on which publisher pages. Advertisers set initial parameters for desirable keywords and budget, and each ad placement ranks the ads according to these criteria, then auctions off the ad space. From Google’s own blog:

> Once an ad is in the AdWords system, it competes against other ads to appear on relevant pages in the AdSense network. It’s through this real-time dynamic auction system that ads are ranked. Those ads with the highest Ad Rank at any given moment are displayed on your site. [9]

Again, publishers, advertisers and Google are all participating in a contract system. Publishers consent to show the ads, advertisers agree to pay for ads placed by Google’s code, and Google promises to pay the publishers. There is a high-level contract that is an ongoing private agreement between these parties to fulfill their roles, and in a sense each ad placement auction is itself a nearly instantaneous, automated micro-contract among advertisers, with Google’s code proxying for all parties.
Human Computation

In 2005 Amazon opened a radical new service, the Mechanical Turk\textsuperscript{8} as an online marketplace and set of APIs for hiring workers to do small jobs, known as HITs. Certain tasks like photo analysis, transcription, or opinion polls are still not possible with computers. The Mechanical Turk addresses these challenges by contracting out the work to people instead of using machines, an approach dubbed "artificial artificial intelligence."

Mechanical Turk HITs are sometimes volunteer, as with the early 2007 search for Jim Gray\textsuperscript{9}, but more often are performed in exchange for micropayments ranging from $.01 to a few dollars. HITs may be posted by requesters via an API, including stipulation of payment, deadlines, instructions, media files, and other criteria. Responses from workers are then submitted back to the requesting service, which may evaluate and rate the accuracy of the worker, affecting reputation for future HITs.

The parallels to Promiserver are obvious, with Amazon essentially having instituted its own, limited-scope micro-contract system. While it lacks explicit programmability, and is geared only towards many individual bilateral contracts, its rich API makes it easy to integrate into other systems, and its use of worker reputation—however simplistic and non-social—is a very relevant data point for systems like Promiserver.

Yet Amazon's contracts are only one way, with distinction between requesters and workers. Promiserver requires no such distinction or specification of commitments. It is hoped that this structural agnosticism will support a more democratic, peer-to-peer promise institution over the top-down model imposed by the Mechanical Turk.

Previous Media Lab researcher William Kelly Norton touched on this more humane approach to human computation significantly in his 2006 thesis *Finishing Touches*. Norton cites his research as part of a larger agenda to create computing infrastructures that are powerful from a traditional computational sense, but also able to leverage the

\textsuperscript{8} Amazon Mechanical Turk - http://www.mturk.com/
\textsuperscript{9} Tenacious Search - http://openphi.net/tenacious/
creativity of human invention. Ideally, the system proposed for this thesis project will be intimately linked to a community of creative labor - humans willing to solve fine-grained design solutions as part of an economy of small scale projects [25, page 38].

Norton’s thesis focuses less on harnessing the intelligence made possible by human computation, and more on the creative potential. Rather than creating jobs performed by remote, anonymous workers, his research strives for a model of human computation that integrates human business, social and creative collaborations. These interpersonal connections are the basic unit of Promiserver.

Community Systems

Promiserver is not intended to foster a new group of promise writers and signers. Rather, it offers a service that can cut across diverse online communities and markets, and a vision of a unifying system for identifying and connecting the interests of these communities via efficient, open commitments. Henry Jenkins describes YouTube’s rapid, massive growth as the result of many individuals directly contributing to their own communities, and in doing so indirectly mixing and learning from one another across community boundaries.

YouTube functions as a meeting place for different subcultures, fan communities, and other forms of participatory culture, enabling the crosspollination of formal practices, themes, and ideas. I see this crosspollination as likely to accelerate the speed with which cultural innovations get picked up and deployed at other social sites [19].

As a model, Promiserver is perhaps analogous to YouTube in this function as a unified point of contact for many communities and markets. It provides a space in which these groups may comingle, critique and learn from one another, borrow techniques and code, and cumulatively form a new form of social promise institution.
2.4 Collaboration

Heralded by Engelbart’s now-classic NLS (oNLine System), collaborative work has long been a core area of research in Computer Science. In recent years much of this research has generally fallen under the heading of Computer Supported Collaborative Work (CSCW) within Human Computer Interaction (HCI).

Among Engelbart’s many contributions, his vision of fluid, adaptable, networked, group-oriented computer systems is perhaps the most prescient and far-reaching, yet ironically continues to be one of the most difficult to bring to satisfying fruition. While underlying technologies have made great strides, many challenges remain in fostering dynamic, creative group work online. Development of the space has been slow and halting.

2.4.1 Coordinator

Spurred in part by these challenges, in the mid-1980’s researchers Terry Winograd and Fernando Flores embarked on a reevaluation of the core assumptions of collaborative software, emerging with a definitive new design imperative for collaborative systems [18]. “The more urgent need,” they asserted, “is to understand the role of background and language in setting the dimensions of the space in which people interpret and generate their possibilities for action” [13, page 156].

Breaking away from the rationalistic traditions of computer science, they turned to alternative approaches in linguistics and philosophy, invoking the philosophical traditions of Heidegger and Godamer to emphasize the criticality of designing for social experience, with language as the foundation of human social activity. The deepest exploration of their approach is their 1986 book Understanding Computers and Cognition.

Meaning is fundamentally social and cannot be reduced to the meaning-giving activity of individual subjects. The rationalistic view of cognition is individual-centered. We look at language by studying the characteristics of an individual language
learner or language user, and at reasoning by describing the activity of an individuals deduction process. Heidegger argues that this is an inappropriate starting point—that we must take social activity as the ultimate foundation of intelligibility, and even of existence. A person is not an individual subject or ego, but a manifestation of Dasein [the essential nature of being] within a space of possibilities, situated within a world and within a tradition [35, page 33].

This work marked a radical shift from rationalistic software—focusing on logical routines in abstraction layers, communicating with one another with the individual user as one endpoint—to social, group-centered design. Given this social language emphasis of the work, it is no surprise that they started with speech act theory (defined page 23), from which they extracted a language/action perspective method of design, characterizing group processes by their fundamental social and language patterns. Individual speech acts became the building block of conversations, which in turn give rise to group action.

This shift marked more than a rethinking of design strategies; it was a change in fundamental philosophy about what it means to create software, and was a milestone in the development of social software and collaboration. The test of this philosophy, and in some sense its posterchild, was their project known as Coordinator, a tool for facilitating coordinated action in groups.

Coordinator decomposed the group collaboration process into atomic elements of language acts, organized into conversations. Each language act was in a sense comparable to an email message, but within Coordinator it was fitted with structured information about the type of language act (directives/requests, commissives, etc), the recipients, and the domain/subject. The structure provides context for the messages themselves, which simply remain unstructured text.

The key contribution of this work in general, and Coordinator specifically, is the flipping of work and collaboration from rationalistic to social and humanist. Coordinator does so through language acts, and to some extent Promiserver follows this tradition. However Promiserver is more deeply rooted in the populist, bazaar-style, even capitalist traditions prevalent on
the web today. Making commitments in Promiserver is a public act in a community and even in the marketplace. This context is the key difference between the two projects.

2.4.2 Alternate Reality Gaming

From this populist web tradition a new style of massive-scale collaborative systems is taking shape in the form of alternate reality games (ARGs). Offering large scale tasks and puzzles requiring players to commit, self-mobilize, and coordinate their contributions, these games can engage millions of people, and bridge the gap between communication in virtual space and collective action in real space.

In describing how these games may lead to new forms of collective intelligence, ARG designer and theorist Jane McGonigal quotes a teacher character from author Vernor Vinge’s novel Rainbow’s End. Vinge’s book tells the story of a future in which connecting technologies have catalyzed a new paradigm of massive-scale collaboration. This teacher advises students to develop their individual talents:

“I have a theory of life...and it is straight out of gaming: There is always an angle. You, each of you, have some special wild cards. Play with them. Find out what makes you different and better. Because it is there, if only you can find it. And once you do, you'll be able to contribute answers to others and others will be willing to contribute back to you. In short, synthetic serendipity doesn't just happen. By golly, you must create it.” ([33, page 60], as quoted by [23, page 4])

The key take in this, McGonigal asserts, is this idea of each person contributing her own unique strengths to larger scale problems. Diverse skills and knowledge prove more effective in achieving collective results, and so distinctive abilities and knowledge are more attractive in the marketplace:

Vinge’s futuristic class therefore offers the students differentiation as a practical strategy for developing individual relevance and
power in a CI [Collective Intelligence] culture. Specialized, distinctive capabilities and resources will later serve as their personal currency in the intelligence market. [23, page 4]

This prediction of an “intelligence market” valuing niche skills, interests, and knowledge begins to sound remarkably similar to the long tail market for services (see 2.3.2). We can imagine these democratized, distributed players dynamically coordinating action via lightweight micro-contracts.

Though work in collective intelligence and large-scale reality gaming is still in its infancy, it is already clear that these games require a core set of committed players. This commitment to the greater game goal—and the creation and adoption of necessary roles and responsibilities—is a form of specialized, nested social contracts within the larger social contract.

### 2.5 Summary

Starting from definitions, we reviewed types of commitments, including promises and variations of legally binding contracts, as well as a new form of commitment I refer to as the “micro-contract”, which is the basis of Promiserver. We also discussed social contract theory and contractarianism—which define the social contract as the base of society and government—and we presented speech act theory as a framework for understanding classes of commitments in language and conversation. Finally, we reviewed a theory for trust based on the economics of promise institutions, showing how it is to an individual’s advantage to work cooperatively. Promiserver aims to build trust in the social promise institution within its online community.

We then covered Lessig’s four regulators—law, social norms, market and architecture—with examples from Second Life of the architecture (code) replacing other regulators. We also looked at Postrel’s dynamist rules, a core set of axioms underlying dynamic communities and governments. Both of these approaches provided conceptual grounding for Promiserver’s use of user created, community enforced contracts, which are essentially mini, dynamic architectures.
The section on commerce looked at the rise of eLancing and the long-tail of niche production and services, all evidence of a distributed, democratized future workforce. It is this expanding economy that is the strongest evidence of need for a lightweight contract system like Promiserver, which would allow individuals and small businesses to leverage one another's specialized skills with faster, more efficient and more dynamic collaborative relationships. We then discussed examples of the emerging industries of trust, including payment services, auctions, lending, human computation, and community systems. Each uses a hard-coded form of commitment; Promiserver factors out these various commitments into a generalized approach to trusted relationships, using code to express the commitment logic.

Speech acts resurfaced in our discussion of the Coordinator groupware system. It was this underlying theory of language actions in conversation that formed the basis of the Coordinator's collaboration process. Promiserver shares this social, structured conversation-style approach, albeit blended into a tradition of open source software and bottom-up, market-based organization. This bottom-up approach to collaboration and commitment is the hallmark of Alternate Reality Games. The massive yet dynamic nature of these emerging ARG communities portends future requirements for fluid, layered systems of social agreements.
Chapter 3

Design

This chapter describes the creation of Promiserver, including my preceding projects that informed conceptualization, as well as previous versions in which I learned lessons about its design that I have applied in this latest version.

3.1 Early Experiments

Starting in 2005 I joined my research group’s agenda to build a community system for collaborative and creative work. This culminated with the release of OPENSTUDIO, an online community and microeconomy for artists.

With OPENSTUDIO, it quickly became apparent that our models for social and commercial relationship were simplistic, and that real human collaborative relationships are stateful, nuanced, and—most importantly—rooted in trust. I began to feel that traditional social network and community systems concentrated on building a quantity of links, but did pay enough attention to the nature of individual, atomic relationships, which in and of themselves are quite dynamic. At the same time, I was hesitant to build processes, rules and regulations into the system itself, and wished to let the community create their own processes.
In looking to create more sophisticated tools for capturing these relationships, I began to explore systems of reputation and trust, including social tagging and critique, audit trails and open transaction feeds, relationships of trust and reputation, which eventually led to social contracts. This section will review these explorations and experiments.

### 3.1.1 OPENSTUDIO

The concept of micro-contracts comes out of my involvement in OPENSTUDIO\(^1\), an online community of people who use a free, simple tool to create and sell drawings in a virtual economy. We created OPENSTUDIO in Fall 2005 as an online social system and infrastructure, with the goal of exploring and prototyping a democratic, free agent creative economy.

The project was a group effort, with each of the original creators bringing in their own unique interests and background, and each eventually spinning their thesis work off of the questions and ideas that captivated them in OPENSTUDIO. We’ll take a look at a few key elements of OPENSTUDIO that motivated this thesis.

**Artsonomy**

The first release of OPENSTUDIO lacked any feedback or communication tools other than the simple actions of buying or selling. While the price mechanism proved a surprisingly effective form of communication, it didn’t capture all observer’s opinions about the piece. In response, we soon added a simple tagging system—dubbed the “Artsonomy” (a play on the traditional folksonomy\(^2\)—for members to categorize pieces.

As expected, we saw simple categorization tags like *red*, *sad*, and *abstract*. However as the community went through different phases we also began to see social commentary, like *ripoff* for copies and forgeries, *money laundering* for pieces transferred among fake accounts, and *expensive* for pieces that were, well, too expensive. The community began to police itself through this

---

\(^1\) OPENSTUDIO - http://openstudio.media.mit.edu/

\(^2\) A folksonomy is a folk taxonomy, a user generated categorization system.
simple mechanism. This proved effective because of the bubble-up nature of the folksonomy; tags applied to drawings become associated also to the people who create or purchase those drawings. Purchasing a drawing indirectly includes purchase of its tag set, and the transfer of these tags is reflected in the profiles of the buyer and seller. In OPENSTUDIO, tags are "semantic lint," sticking to pieces and the people in contact with those pieces.

Figure 3-1: OPENSTUDIO's Artsonomy is not only a classification system, but also a means of socially commenting on the work. When tagging a piece, users are shown which other members tagged similarly (left). These tags then bubble-up to the tag cloud of the users (right).

**Transparent Processes**

Throughout the development of OPENSTUDIO, we were consistently faced with choices about which information should be private versus public. Created documents, purchased documents, account balance, transaction history—all were debated. In the final design we kept much of the data radically open, including traditionally sensitive information like bank account balances and transaction histories.
Figure 3-2: OPENSTUDIO featured numerous open views of historic data, including account status and history (left), provenance of the piece (middle) and change history of the piece (right).

The result is a rich audit trail of activity, with RSS feeds for recent creations and purchases, as well as provenance and change history for individual pieces. Combined with the Artsonomy, these processes revealed and further enhanced the creative and economic processes of participants.

**Hire-a-Designer**

Hire-a-Designer was implemented by fellow researcher Kate Hollenbach. Early versions were embedded directly into a lightweight spreadsheet program, providing users with an interface to search for and hire designers to create graphs and visualizations of their data. This prototype was part of a larger, group-wide vision of building applications that take advantage of community networks and human creative abilities.

Later integrated into OPENSTUDIO, hire-a-designer evolved into an artist commissioning feature, in which the commissioner/client requests a piece created according to certain instructions, for a particular price, and optionally with some deadline (Figure 3-4). Artists accept jobs that appeal
to them, and clients can cancel jobs that miss the deadline. Payment for the service goes into “escrow,” meaning that it is deducted from the client at the time of the request, and redeemed by the client if cancelled, or by the artist if completed.

While this process was fascinating, it quickly became apparent that the rules of these commitments were governed quite strictly by the architecture of the system. I began looking for a way to abstract out this commitment logic from the rest of the code in a manner that would allow participants to define their own working relationships.
Figure 3-4: The Hire-a-Designer feature in OPENSTUDIO. Artists may make themselves available for hire (left). All offered, current, and past jobs are all published (right).

3.1.2 Organic Marketing & Consumer Trust

In January 2006, the Media Lab Simplicity Consortium hosted a 24 hour prototype-a-thon competition themed around “Organic Marketing.” The challenge was to produce a prototype of some advertising system to foster trust and truthfulness in advertising. The submissions varied, but the majority of teams focused on the use of meaningful consumer participation to create trusting relationships between consumers and advertisers.

Months later, in preparation for a followup meeting with lab sponsors Time-Warner and Johnson & Johnson, I had a chance to revamp and implement one of the winning prototypes. The system, dubbed Clickback, provided a simple interface for viewers to positively or negatively review an advertisement, with negative votes result in a new ad showing up. See Figure 3-5.
Figure 3-5: Clickback provides a mechanism for consumers to participate in the advertisement process, as well as view the statistics of the advertisements.

The results are stored on the ad server, which keeps track of which ads are popular on a per-publisher basis. Negative ratings make ads less likely to appear again for that publisher, while positive ratings reinforce the ads for that publisher. I also added a “flip” interface feature, which reveals a standard view showing the statistics on the advertisement, as well as the url it will bring people to.

The underlying idea with Clickback, as well as other projects that came out of the contest, is that consumers understanding and participating in the process of selecting ads will trust the publishers and advertisers. Clickback parallels a larger initiative being pioneered by the lab and its media company sponsors known as OpenBrand, which aims to create a social contract among participating publishers and advertisers to open themselves to consumer feedback and critique. In this model, with the possibility of negative feedback and tarnished reputation, advertisements become more like commitments or declarations than simple persuasive instruments. This public, transparent commitment process is very similar to the principles explored in
3.2 Prototypes

3.2.1 Natural Language

The first version of Promiserver was a simple, bare-bones interface with a text form for writing the terms of the promise in natural language. Roles in the promise could be specified with special syntax within the plain text. Participants could then assume roles and sign them.

Figure 3-6: Prototype v1: Natural Language
Social contracts require that the public be able to witness and comment on the promise. This is similar to the SEC rules governing material contracts in publicly traded companies. Other than this sort of exceptions, however, legal contracts may remain private unless brought to court as part of a lawsuit[5].

As a social promise institution, this first version of Promiserver published signed promises on the front page. This experiment, minimal as it was, effectively revealed the power of social commitments in an online community. Given the proper forum, publishing a commitment online has potentially much stronger social ramifications than a simple signed piece of paper.

Yet it was not the departure from traditional contracts that I was seeking. For the next version I radically shifted to use of a programming language to describe the contract conditions.

3.2.2 Code and Modularity

As good software design builds a modular system of layered abstractions, so too it seems like commitment logic should follow the same pattern, allowing composition of basic components to create more sophisticated systems. This would entail integrating multiple contractual elements, building systems of commitments with contingencies on other commitments, and creating de facto community standards for certain patterns and clauses.

While certain branches of law use standardized contracts to streamline transactions, much of the legal system still makes use of custom, single-use contracts. In working with intellectual property and corporate law attorney Brent Britton, I was surprised to hear that while deals may start with a template, the contracts often involve rewrites [5].

Hoping to bridge this schism, I created a draft version that used coded clauses that could be reordered within the contract, similar to some email filtering systems. The eventual goal was to allow copying and reuse of individual clauses, creating a library of micro-contract logic snippets.
While novel, the interface proved clunky and toy-like, while still trying to do too much all at once. It also distracted from the core idea of programming a contract, turning the process into one of drag and drop over working with a new type of language. I retained certain elements for the final version, such as live updating and error checking, but abandoned the draggable, modular clause concept.

### 3.3 Design

The earlier work and predecessors proved useful in crystalizing the exact design of Promiserver. In this section we will outline the design goals of the system. This section presents the fundamental criteria for the system, as well
as conceptual design of promises and the roles and interactions of participants.

3.3.1 Criteria

Based on previous work and the lessons learned, Promiserver’s design necessarily faces certain requirements.

- **Promises are written in code.** The system provides some interface to aid authoring and debugging.

- **Promises maintain state.** Promise state reflects the status of the real, human relationship. During creation, promise authors test different states to debug the process before signing. State includes current time, which is always changing.

- **Promises are architecturally binding, but not architecturally remedied.** The commitment is built into the architecture, yet the meaning of the commitment lies with the members of the community. Enforcement/remedy should be a community process that utilizes social elements such as reputation, guilt, shame, or honor. Different sub-communities can create their own enforcement rules.

- **All processes are transparent and auditable.** Witnesses and participants should be able to see not only the outcome of the promise, but why the promise turned out the way it did. Transparency and openness are key ingredients in dynamic systems.

- **Promises are accessible and manipulable programmatically.** The intent of Promiserver is eventual integration with other systems, not a standalone service. Systems that rely on trusting relationships should be able to use Promiserver via an API, to create promises for people to sign, or even to sign and monitor promises on behalf of people.
3.3.2 Anatomy of a Promise

Following the criteria above, Promiserver's promises are virtual objects composed of five ingredients:

- Metadata such as title, description, tags, comments, creator, and overall status (draft, published, signed, cancelled, breach, or success).

- One or more participants who must sign the promise, and the outcome of which reflects on them.

- A bundle of named variables and associated values that represent the promise's current state.

- A chunk of code that will be evaluated with respect to a binding derived from those variable values.

- An audit trail of history of changes to the promises state.

The status of the promise changes as it progresses through its lifecycle.

Figure 3-8: Promise Lifecycle.
3.3.3 Implicit Roles

Within the target audience for this system, we distinguish between three essential roles: authors, signees, and witnesses. These roles are simply descriptive, and are not built explicitly into the system. A user may switch between any or all of these roles in a single session.

Authors

An author writes a promise, either for her own participation or for others to sign. For simplicity, we assume that authors know which people will be participating in the promise. The author’s process is approximately as follows:

1. Log in to the system.
2. Select the option to create a new promise. Title it and provide metadata such as description or tags.
3. Write the code. This involves writing computer code into a text box, and receiving real-time visual feedback on the input. Syntax and some run-time errors are reported, and free variables are parsed out to represent the promise state. The initial values of these variables can then be set, also resulting in feedback.
4. Assign participants. Search or browse for users within the system, add them to the promise, and label them with their roles. A participant may fill multiple roles in the same promise. An eventual extension may be to allow groups of participants to assume roles.
5. Publish. The only requirements for publishing are that promise code is well formed, does not begin with a success or breach, and that there are one or more participants. Once published, participants are notified with a signature request.

The author needn’t be a participant in the promises she creates, however she remains associated with those promises, including their outcomes. This author accountability is intended to incentivize careful promise crafting.
Signees

People are bound to the promises that they sign, and transitively to the other people that sign it. A signee participates in the promise, and may edit its state.

1. Receive email invitation to sign a promise.

2. Log in and view the promise and the reputations of the other participants.

3. Sign or decline accordingly. Optionally tag or write a comment.

4. Wait for all other participants to sign, at which point the promise becomes active. Active promises are continually re-evaluated, and the results are logged and published as part of an audit trail.

5. Once active, optionally request cancellation. If all signees also request cancellation, the promise is cancelled.

6. When necessary, modify the values of the promise state variables to reflect changes in the relationship. Review the effect of these changes, and commit or cancel. Changes are logged and published in the audit trail.

7. State changes, passage of time, or other factors in the code may lead to a breach or success condition. The promise is marked accordingly. Comments/tags remain open.

A signee is engaged with her promises and the other signees, taking time to understand the code, repeatedly visiting her promises to monitor their states, and conversing with other participants.

Witnesses

All published promises are publicly viewable, even to non-authenticated visitors. A witness is a visitor who looks at the promise. If logged in, a witness may optionally tag or comment on the promise.
1. Browse or search the website.

2. Examine promises, including participants, code, state, audit trail, and comments/discussion.

3. If authenticated, leave a comment or tag.

Witnesses can come and go, sometimes leaving their mark on the promise via comments or tags. Though the role itself is minimal, all viewers of the site are witnesses, and contribute to the overall strength of the Promiserver's social promise institution.

3.4 Implementation

This section describes the implementation of Promiserver, including underlying infrastructure technology, as well as design and construction of distinctive features grouped into two classes: community processes and contract language.

3.4.1 Infrastructure

Promiserver is a web application, compatible with modern browsers, with some data and functions also available via a REST-style API. The interface is entirely XHTML and CSS, making extensive use of Javascript for responsive interaction.

On the back end, the system currently runs on a dual-core Xeon rack server running Red Hat Enterprise Linux with a RAID 1 storage system and 16GB of memory. I set up this system as the primary deployment machine for our research group in Summer 2006. All of our software runs on a stable, production-level open source stack, with Apache 2.2 for the front-end webservice, running a load-balanced proxy to a cluster of Ruby on Rails processes, with data persistence in a MySQL 5 relational database.
3.4.2 Community Processes

Independent of the form of the contracts, one of the main conceptual points of Promiserver is to provide architecture that supports a social promise institution. With this in mind, and drawing from research and experiences with open architectures and audit trails, I added certain key features to Promiserver.

Transparency

While draft promises are private to the author, all promises must be published in order to be signed, opening them up to critical review not only by prospective participants, but the public as a whole. If the promise is signed by all participants and becomes active, each participant has equal rights to modification of the promise state variables, yet each of their changes is logged in the publicly viewable audit trail. This history feature is similar to the workings of wikis, forcing accountability for all participants’ actions.
While strictly private promises were a considered feature, they become extremely difficult to enforce. One possible extension would be promises that remain private unless a breach occurs, at which point it becomes opened to public scrutiny and comment.

**Accountability**

The state or outcome of each promise is architecturally tied to its participants. Each participant’s profile contains a list of all promises in which she has participated, as well as a set of connections forged via those promises. Promises authored are stored under a separate tab. See Figure 3-10 for an example.

A later iteration of Promiserver also introduced small bar graphs next to each user’s icon. Each graph indicates the number of some promises in particular state that are connected to that user. The intention with this design is to publicly reveal overall tendencies for certain behavior, such as authoring versus participating, or successful completion versus breaches. An example of these mini-graphs can be seen in Figure 3-11.

**Comments**

Promises are designed to be endpoints not only for microcontracts, but more importantly for mediation around those promises. All promises may be commented on by the wider Promiserver community. While early versions allowed access to comments via links, based on early user feedback the comments were moved to the front page, encouraging participation in the dialog. See Figure 3-12.

**3.4.3 Contract Language**

The second main unique component of Promiserver is its use of code to represent commitments. Use of code presents many interesting challenges, technical as well as design. This section will take a closer look at the contract
Figure 3-10: Profile Page. The profile page shows the promise participation history and statuses (center), as well as aggregated connections to other participants and the outcomes of those connections (left).

coding features, including the language itself, the interface design choices, and the technical challenges.

Ruby

Participants write their promises in code, specifically in Ruby. Ruby is a popular high-level object-oriented scripting language with notable functional-programming qualities. It has been well suited as the underlying Promiserver code for a number of reasons.
First, Promiserver itself is written in Ruby, making it possible to evaluate the promise code with respect to custom variable types, data and methods. Second, Ruby scripts are evaluated at runtime, with pre-checks for syntax errors, and capabilities for handling runtime errors. This makes it possible to do fast, live parsing and evaluation without compilation. Errors can be quickly reported back to the author.

Additionally, Ruby’s flexible syntax and metaprogramming features encourage creation and mixing of domain specific language (DSLs). Promiserver’s DSL is very simple, with reserved functions success and breach, each optionally taking a description string. Consider the following program, an early promise created on the alpha version of Promiserver:
Figure 3-12: Participation. Promises offer a forum for discussion and critique, or even just fun. Any logged in participant may comment on any promise.

```ruby
finish_by = Time.parse "4/22/2007"
if Time.now > finish_by
  if can_change_folders
    success "new feature finished"
  else
    breach "didn’t finish new feature by deadline"
  end
end
```

The lack of extra syntax makes this program highly readable, in some parts bordering on natural language. Ruby is lenient in its syntax requirements,
with features such as optional parentheses and end-of-line semicolons, as well as flexible whitespace.

**Fast, Browser-based Evaluation**

Promiserver’s programming environment is designed to facilitate the writing and debugging of promise code. As the user types, a javascript handler asynchronously submits the code to the server via a Representational State Transfer (REST) API. The server parses the code, checks for syntax errors, extracts unbound variables, evaluates, and returns the results in Javascript Object Notation (JSON) format, where they are displayed visually to the author. Running on the production system over a broadband or better connection, the delay is negligible.

![Promiserver's authoring mode](image)

Figure 3-13: Promiserver’s authoring mode performs fast, live parsing on the code while the user types, checking for syntax errors, extracting variables, and reporting on breach or success conditions.
The end result is that promise authors are immediately alerted in the event of syntax errors, certain runtime errors, or breach or success conditions. Breaches and successes also highlight the line on which the ending action occurred. A separate monitoring process invokes the same evaluation routine to check the status of all signed, active promises at a regular interval. All evaluation results are added to the audit trail.

**Variable Extraction**

All code processed by the debugger and evaluation system is subjected to a parsing process that converts it to a parse tree composed of S-expressions, represented as arrays. This parse tree is created using the ParseTree⁴ open source Ruby library. During promise authoring, these resulting S-expression are scanned for free variables, which are then extracted and used to instantiate variable objects associated with the promise, as well as being returned to the user in the web page interface.

**Security**

Running user code on a server poses a major potential security risk, and requires extreme caution. Promiserver is written in Ruby, and calls eval on promise code within a sandboxed environment⁴ in a thread, automatically cutting the process short if it takes too long or recurses too deep, such as in Figure 3-14.

In addition, the entire promise creation workflow is also subject to a security process, with filters running on each request to contextually determining privileges depending on the user’s session as well as the state of the promise.

---

⁴ The sandbox used by Promiserver is Ruby’s built-in SAFE variable mechanism. Future version of Ruby will integrate a more sophisticated Sandbox mechanism (http://code.whycl.setEmail-kystiff.net/sandbox/).

Figure 3-14: An example of Promiserver security. This user’s program is an infinite recursion, but in the sandboxed environment the exception is caught.

**Constant Injection**

Contracts and promises are typically commitments to future states or actions, dealing with issues of time, such as deadlines or start dates. Ruby’s Time library provides functions for accessing the current time (using Time.now) and comparing times (via overloaded numerical comparison operators <, <=, ==, >=, >). Signed, active promises are constantly reevaluated, so the Time.now method returns the system time on each evaluation, making possible time-dependent conditions. Consider this example promise that has an official deadline, as well as an unofficial three day extension, both hard-coded.

```
official_deadline = Time.parse "5/11/2007"
```
unofficial_extension = 3.days  
if Time.now > official_deadline  
  if Time.now > (official_deadline + unofficial_extension)  
    breach "you missed the deadline"  
  elsif is_complete  
    success "lucky you finished it within the extension!"  
  end  
elsif is_complete  
  success "You finished it!"  
end

These time-dependent promises will be difficult for users to test since they depend on logic that occurs only in the future. Anticipating such a need, Promiserver’s evaluation library includes an optional parameter to pre-specify the current time context, i.e. the value of Time.now. Internally, the system includes a mixing in an additional method to the Time class that creates a clone Time class with overridden Time.now and Time.new methods. It then uses a technique known as constant injection[24] to reassign the Time constant this new, dynamically created class within the scope of the evaluation. See appendix, page A.1. This feature is not utilized in the current interface, but in the future may allow for timeline or calendar views of the promise evaluation.
Chapter 4

Evaluation

This chapter reflects on Promiserver by looking at both user experience and conceptual context. The user experience analysis is grounded in findings from critical reviews and workshops with early participants.

The concept evaluation applies theories presented in the Background section, assessing Promiserver’s relation to existing work. These findings pave the way for the final chapter’s focus on lessons learned and future work.

4.1 Usage

User study and evaluation is of critical importance to a socially oriented project like Promiserver. While original plans called for a public rollout and usage to gather data and feedback, as with many software projects the design and development processes took longer than expected, delaying public release.

In March I began to show the first working versions, starting internally with demonstrations to visitors, and later a workshop on Promiserver for the Simplicity Consortium event. In mid-April I finally published a live alpha version at http://promise.media.mit.edu/, and announced it on blogs, to the local research community, and several mailing lists. As of this writing promises have begun to show up in the system. There are currently 18 users,
and 36 published promises. Of these published promises, 5 have been
d eclined, and 26 have been signed, with 5 as yet unsigned. 19 of the signed
promises are complete, with 12 successes, 5 breaches, and 2
cancellations.

This section summarizes the early usage and various critiques to the system.
I have conducted ten informal interviews with participants, reviewing all
aspects of the system as it stands, and discussing usage scenarios and
possible future work. Two of these results came in via email and chat
sessions, but the majority of the reviews were live, informal interviews
conducted in person. Additionally, some users have posted comments within
the system itself, or emailed me regarding bugs or confusion, and these issues
are also integrated into this evaluation.

4.1.1 Information Architecture

Informally interviewing reviewers while they used Promiserver, all found the
interface overall fairly manageable, and were able to login, view their profile,
browse people, and start a new promise. Certain elements of the interface
were points of confusion, due to technical or design issues, and will need to
be reworked.

Too many tabs

Several reviewers pointed out confusion with the tabs in the interface. The
promise creation tabs represent steps in a process, which differs from the
usual tab metaphor as an non-sequential cluster of related panels, and also
functionally conflicts with the other tab interfaces used in the site. See
Figure 4-1.

Additionally the tab options change between draft versus published promises,
and reviewers found this confusing as well, most likely because tab interfaces
in traditional desktop systems are not modal or contextual. Three reviewers
remarked that the multiple tab-based navigation options made the site look
too busy, and that it was hard to know where they were. At their suggestion, I later changed the top navigation to not use a tab-style appearance.

Auto vs. Manual Save

During promise creation, an ongoing javascript routine continuously monitors user input and automatically relays changes to the server. Typically web interfaces require a confirmation of changes from the user, and certain reviewers were unsure that their data had been saved. One user suggested moving the auto-save status indicator closer to the field changed.

4.1.2 Programming promises

One of the core premises of Promiserver is that code is a viable alternate, possibly preferable form of contract language, offering some sweet spot of objective, debuggable, procedural logic, mixed with input from external variables. As such, the programming activity is intended as one of the primary target areas, and its evaluation is crucial.
All reviewers I interviewed had notable experience programming, and a few had specifically written programs in Ruby. And all were eventually able to write their own promise. Yet observing their approaches and the results, it became clear that the promise programming process is initially very challenging, and that there remains much room for both design improvements and further innovation.

Starting from scratch

Presented with a blank text area, the task of coding a promise is daunting. What does it mean to write code to describe a social process? Almost all reviewers felt confused when first encountering this particular page (Figure 4-2).

![Figure 4-2: Daunting Blank Promise. All reviewers were unsure how to proceed when presented with this screen. Most reviewers found the process initially difficult, though generally warmed to it in subsequent interactions. The use of a programming system](image-url)

The use of a programming system...
immediately and significantly raises the barrier to entry, essentially barring non-programming users from authoring. This barrier to entry is one of the original concerns I had about the programming approach, and is why I initially considered prototypes with modular, draggable clauses (see discussion of the modular clauses prototype, page 55).

In some part their trouble may stem from the foreignness of the core contract-as-code concept. Some reviewers explained that they found it difficult only because it was a new concept, and that it made more sense after the initial hurdle. While these explanations are encouraging, future design iterations must make the promise creation process accessible to a wider range of people. This may require changes to the fundamental programming-based approach, the support tools and documentation, or creation of external systems that create promises via the API.

Watching people write promises, certain code patterns emerged very quickly, such as a consistent reliance on (and confusion with) date comparison, and a simple if...elsif...end form was extremely common. Seeing these patterns emerge, it is tempting to redesign the code itself to make these patterns implicit. Yet in doing so, there is a danger of building unnecessary rules into the system, of unnecessarily restricting Promiserver architecturally.

Finding help and examples

In a first attempt to lower this barrier to entry, I began work on a help section that explains the promise creation process, shown in Figure 4-3. Unfortunately, including a prominent link to this section on the promise programming page proved insufficient, as most subsequent reviewers failed to notice it. When I pointed it out to them, they did not take time to examine it.

Multiple reviewers independently stated that they would prefer inline examples, and options to load common promise example patterns. Most all users found it helpful to look at others’ promise code before writing their own, copying and pasting the code and modifying it to suit their needs. Some
How do I write a promise?

There are two reserved methods: breach and success. Each takes an optional message parameter. Your code defines the conditions under which the promise breaches or succeeds. For example, a really simple promise would be:

```plaintext
if condition_x
  success "it happened"
end
```

The value of the condition_x variable determines whether the promise evaluates to success. When condition_x evaluates to true, the promise will be evaluated as a success, with corresponding message "it happened". A marginally more sophisticated promise might look like this:

```plaintext
if condition_x
  success "it happened"
elsif condition_y
  breach "sorry, y happened"
end
```

In this one we have possibility of a breach as well, if condition_y evaluates to true.

Promise State

Those variables like condition_x and condition_y aren't defined anywhere in this code. This means that they are free variables. Free variables pop up to the left of the code text area as you type them.

Together these variables form the state of the promise. When writing the promise, you can change the values of these variables to the desired initial state. Once published and signed, all participants can also change the values of these variables.

Time

Most promises are commitments with respect to some future action or event. So it may make sense to take time into account.

```plaintext
if document_received
  success "thanks, nice work"
elsif Time.now > document_deadline
  breach "missed the deadline"
end
```

Again, document_deadline and document_received become promise state variables, meaning they can be changed by the participants. If we wanted to fix the deadline and remove it from the state, we could simply add it in a line.

```plaintext
document_deadline = Time.parse("5/11 2007 6:00pm")
if document_received
  success "thanks, nice work"
elsif Time.now > document_deadline
  breach "missed the deadline"
end
```

Figure 4-3: Getting Help. The help/faq section was expanded after several reviewers found the promise programming interface confusing.

Others requested being able to explicitly use another promise as a starting point. Others asked for a more comprehensive help system within the page. One reviewer also suggested color coding variables and special keywords.
State variables confusion

During the programming process, free variables are automatically extracted from the code and pulled out to the left, into a box labeled “Initial State” where the values may be manipulated (Figure 4-4). This parsing trick was intended to ease development of promise code by highlighting for authors which variables could act as inputs.

![Figure 4-4: State Variable Confusion. State variables extracted on the left were sometimes initially interpreted by users as displays of current values rather than controls.](image)

Reviewers were split on this design. Some seemed to grasp it immediately, understanding that these variables could be assigned values, and playing with the values to test out their code. Others did not understand how to use it, even after I tried to explain it. One reviewer described thinking that these variables and their values represented a dashboard summary of the code, and was frustrated that the values did not change when he assigned them values in the code area.
Timing

Working with time often proves to be one of the more difficult aspects of promise programming. While many programmers deal with parsing and displaying times, it is unusual to think in terms of logic and dependencies involving hours, days or weeks. Having set up their promises to mark a success or breach on some future date, we each ask a very simple and important question: How do I know it will work?

This is of course a fundamental, philosophical problem in computer science, the Halting Problem, a theory stating that it is impossible create a generalized system for predicting completion of Turing-complete code. Yet in crafting commitments this problem is even sharper, as we are forced to look at code that executes only in the future and not now, and we must trust that this code—and the system that runs it—will behave the way we intend it.

Computers also handle time a little differently than people. One reviewer’s promise uncovered an unintuitive element of date handling, with Ruby’s parsing of a date without a time—such as `Time.parse('5/11/2007')`—resulting in a time at the exact beginning of the day. This implicit assumption within the interpreter results in an unexpected breach. Supposing the time is, for example, May 11, 2007 sometime in the afternoon, this chunk of code would unintuitively result in a breach.

```ruby
if Time.now > Time.parse('5/11/2007')
  breach "you missed the deadline of May 11"
end
```

So programming the logic of future events is tricky. There are at least two possible approaches to helping promise programmers better understand timing. In the near term, we need an interface to allow testing of alternate values of `Time.now` using the constant injection feature already included in the promise evaluation system (page 69). This interface could take the form of a date entry widget, a timeline, or even a calendar.

A longer term project, far beyond the scope of this thesis, would be a redefinition of the language itself in a manner that allows some symbolic
analysis of the code to attempt to determine which time dependencies will be triggered and when.

4.1.3 The Role of Roles

Currently each participant takes part in the promise via one or more labeled roles. These promise roles are intended to help explain the promise—and the person’s relation to it—in non-code terms, making possible eventual features such as search, aggregation, or organization by role. They may also eventually be used to enable people to easily make derivative promises, keeping the same roles but substituting in new participants. The extra layer of abstraction makes it possible to author promises without any particular participants in mind.

Users, however, were at times unsure of how to use the roles. Additionally, these roles may include informal semantic information that potentially competes with or distracts from the actual promise code. If roles are to be used it is important to make their value more apparent.

One reviewer proposed a notable extension—one which I had independently considered but had ruled out as overly complex given the timeframe. This proposal calls for roles to be referenced as objects within the promise code, with variables and breach/success statements optionally scoped to these role objects. This extension would tie the roles more directly into the code, adding expressivity to the promise language by granting access to state variables only depending on certain permission. Consider the client/contractor promise with this hypothetical addition to the coding language:

```ruby
if contractor.receives_payment
  if client.receives_deliverable
    success "everyone wins"
  elsif
    contractor.breach "missed deadline"
  end
elsif Time.now > payment_deadline
  client.breach "didn't pay"
```

79
4.1.4 Authoring & Committing

Does authoring a promise also entail taking on and signing a role? Or does it make sense to create promises only for other people to commit to?

A number of reviewers assumed that authoring a promise was the same as making a promise, i.e. that as author they were implicitly and automatically bound to the promise, and that the other participants were the people they make the promise to. This interpretation was not the intended one, and in fact improperly constrained users to only a subset of possible promises.

For example, one reviewer created her first promise without including herself as a participant, though the commitment was about her own future actions. In a later interview session, she explained that she had thought a promise was transitive, made by the author of the promise to the other participants. While hers was a perfectly understandable interpretation, it was one that the design overlooked. Possible remedies would be to require that the author be a participant in the promise, to devise new terminology for labeling these commitments, or simply to add further instructions that clearly explain the use of participants in these systems.

As a first past remedy of this ambiguity, the promise creation process was revised to optionally target particular users, with more explicit instructions that the promise was with the person, and that participation could be rearranged. See Figure 4-5.

It is hoped that such incremental clarifications can reduce user confusion. Admittedly, however, the project’s adoption of the term promise—both a noun and a verb—does result in a certain ambiguity and possible interpretation of transitivity. In this sense the term contract or microcontract may be preferable.
4.1.5 Enforcement & Remedy

Community systems often take time to develop, if they develop at all. OPENSTUDIO took well over a year before it hit its stride, and as the number of people increased we added new features that were only valuable as the community scaled, such as tagging and improved search.

With its focus on this new and still obscure concept of promise programming, Promiserver is a much more niche project than even OPENSTUDIO. It is also still in the very early stages. Due to this newness and the rapid development timeframe, Promiserver has had limited public usage, and the efficacy of its community enforcement and remedy is, as yet, unproven. So far most contracts have been tests of the system, and the majority of the users are already familiar with one another via social or academic contexts. Real reputations are not at stake.

However some initial reviews indicate that the public declaration of intention
is a motivator. One reviewer created an early promise to finish an element of her research by the end of the week, and stated later that this public, non-retractable commitment did motivate her to finish. Unfortunately the participant designated to sign off on her promise failed to do so, and the promise ended as a breach anyway.

Apart from these anecdotal accounts, as yet there simply is not enough data to know whether these social commitments are enough for people to keep their word. The overhead of coding and debugging promises has taken center stage with the development and review processes. For now, without sufficient data or usage experience, evaluation of the social enforcement system will have to depend on comparison to and framing among existing approaches.

4.2 Concept

We now step back to analyze Promiserver as a concept, starting from its current implementation, and generalizing to hypothesize further about systems of commitment. What are the core propositions of Promiserver, both explicitly stated from the outset, and implicitly wrapped within its architecture?

Based on frameworks reviewed in the background section, and in light of earlier work and experiments as well as Promiserver itself, we may begin to find common axes that contextualize this project and point the way to further work.

4.2.1 Dynamic Architectures

Whether in physical reality or in code, architecture defines space and regulations by which communities form, exist, thrive, or die. As a system of code in which people connect, communicate, and commit to future action, Promiserver is an architecture for facilitating dynamic formations of diverse communities.
Postrel asserts that dynamic societies require a core set of regulations that form a foundation out of which community-specific rules may emerge and evolve. The regulations she proposes—the **dynamist rules** (see section 2.2.3, page 31)—are base layer protections. She uses Stewart Brand’s analogy from *How Buildings Learn*, which classifies the elements of buildings into a spectrum, from fundamental to dynamic: site, structure, skin, services, space plan, stuff. [4, pages 12–23][27, page 143]. Dynamist rules are the site and the foundation that set the stage for more dynamic rules.

**Promiserver** is a community architecture built according to these dynamist rules. Its auditing and open data create accountability and permanence in recording commitments, while it also allows for public critique of commitments. Most importantly, however, is that it is an architecture for what Postrel calls **nested rules**: rules within rules.

> Nested rules...recognize the diversity of human bonds. They protect plenitude—and voluntary community—by allowing individuals to choose the specific rules under which they prefer to be governed. They permit choice, competition, and learning, rather than imposing a single, static model [27, page 145]

Promiserver is an architecture in which people create their own dynamic, nested, lightweight architectures. These nested architectures are promise code. The code begins only as a set of structured language actions, in some sense similar to the Coordinator (2.4.1, page 42) in the process by which it is created and edited, structured data is attached, it is published, then declined or signed.

Once signed things get interesting. Unlike legal contracts or even traditional social commitments, which are human enforced, signed promise code is self-executing. And as a self-executing regulator, it fits well with Lessig’s definition of architecture. Its outcome, assuming it ever halts, will automatically go into the record of all involved participants. So signing a promise essentially transforms it into a nested-architecture.

However, while technically self-executed, Promiserver gives authors and participants the option to allow for other regulators on the code, via the
state variables. Depending on the code, the variables may provide hooks by which participants may influence or even determine the outcome of the promise. So the promise is self-executing, but the input to its execution may be made to depend on human input. The level of autonomy of the promise code is completely determined by the author.

Let's look at two different promise examples representing each end of this spectrum, from agent-executed to self-executed. First, consider a promise in which the variable states completely determine the outcome of the promise:

```ruby
breach("x happened") if x
success("y happened") if y
```

Participants have total control over this promise outcome simply by setting variable values for x or y. Despite being in code, this promise is essentially agent-regulated, and is subject to all the market, social and possibly legal regulations that dictate people's actions. With promises like these, Promiserver acts simply as a forum in which the commitment is posted, and a common database in which the outcome is stored.

Yet it is just as easy to write a promise which executes based on no input variables, that is completely self-executing and beyond the control of its author or participants. The following promise waits for a certain date, then essentially flips a coin to determine whether the outcome is a success (see also “a little random promise” in A.2.3, page 98):

```ruby
if Time.now >= Time.parse("5/12/2007")
  (rand > 0.5) ? success(":"): breach(":")
end
```

Of course, regardless of the specifics of the promise code, the outcomes of the code only regulate future action via human agency. In the end, when promises complete, the only thing these promise architectures regulate is a database, and by extension a website. They alter states and flags on variables, and these variables are associated with the participants.

How much weight we give to these outcomes is totally determined by the
people in the system, by the community. Promiserver displays the outcomes, and provides an audit trail, but it does not regulate people’s actions based on these outcomes. These regulations fall back to Lessig’s other regulators: social norms, the market, and potentially law.

4.2.2 Language & Commitment

Promiserver’s mechanisms for requesting, making and canceling commitments echo some of the elements of speech act theory, in particular Winograd and Flores’s application of speech act theory for groupware systems such as the Coordinator (previously discussed in 2.4.1, page 42). Like their work, Promiserver takes language acts as a starting point, framing the problem of collaboration and coordination in a humanist, language-act tradition over the mechanist, rationalist tradition.

Though Promiserver uses coded commitments in place of natural language, and though the origins of these projects differ, there are remarkable similarities in the overall approaches of the Coordinator and Promiserver. Both offer structured, collaborative, goal-oriented systems with open-ended, user defined topics. Coordinator’s ability to search for conversations by status is also built into Promiserver, with it’s organization by promise status. Winograd and Flores’s example questions “In which conversations is someone waiting for me to do something?” or “In which conversations have I promised to do things?” [13, page 160] both have analogues in Promiserver, and in fact Promiserver uses these criteria as the base organization system for promises. Promiserver also represents which interactions are open versus complete, similar to Coordinator’s “open” versus “complete” statuses [13, page 162], yet expanding on the definition of completion with a structured classification of the outcome as breach, success, cancellation, or declination.

Perhaps most prominently, Promiserver, coming 20 years later, operates in an entirely different tradition that simply did not exist in the 1980’s: the web. As a web-based system, it offers an open tool and forums in which people may decide to participate. Coordinator, on the other hand, is a system intended to be applied top-down in work settings, and users must adopt its architecture. Promiserver is an architecture in which there can exist
a chaotic bazaar of services rather than a monolithic collaboration tool. The collaboration will happen naturally within the open market [29].

Reviewers in fact suggested a further loosening of Promiserver's structured process, removing rules that currently restrict user actions such as publishing already-breached promises, or modifying promises once published. Seeing as imposed structure was one of the criticisms of Coordinator [18], it seems appropriate that Promiserver’s imposed structure may turn out to be one of its shortcomings. Future iterations will relax the workflow to give users greater leeway in their usage of the system.
Chapter 5

Conclusion

This thesis has presented Promiserver, a web-based service and toolset for creation of lightweight contracts—*promises*—in code. Once debugged, published and signed by participants, promise code is evaluated on the server, with outside hooks in the form of state variables that may be modified by participants. The final outcome of the promise—*breach*, *success*, or unanimous *cancellation*—is reflected in the profiles of all participants.

We position Promiserver at the intersection of several traditions. First, Promiserver's core philosophy of modeling relationships as contracts is an idea stemming from the long histories of philosophical, economic and legal contract theory. Second, Promiserver's humanist, language, and commitment based approach to collaborative work recalls the key elements of speech act theory, in particular its later applications to groupware systems such as Coordinator.

Yet the key distinction from the contract theories and the language act approach is Promiserver's thorough entanglement with the web. The web is a sprawling, chaotic mess from which countless communities, rules, social norms, and markets have emerged. Beneath this messiness is an underlying architecture built primarily on an ethos of openness, flatness, and further programmability.

The web, then, is an ultimate example of dynamist architecture, as it makes possible so many varying community rulesets nested within it. Within the
context and tradition of the web, Promiserver is one more nested ruleset, a nested architecture. Promiserver offers people tools to write code for their own architectures, describing the logic of human commitments instead of the logic of the network.

5.1 Lessons & Next Steps

5.1.1 Language & Code

Most prominent among reviewer feedback was the consistent difficulty in understanding and writing promise code. While some users believed that it would become easier with time, this initial barrier to entry was huge. The coding element, while a new and valid early attempt, is simply too complex. As one user described it, people do not want to code commitments, but they do want some way of objectively, visually expressing them. There are a number of possible remedies:

- Removing the coding system altogether, replacing it with natural language commitments, optionally with a few special syntax elements. This would be a return to the original Promiserver prototype (3.2.1).

- Augmenting the coding with a more active, helpful interface. This may include inline help, code completion, templates, syntax highlighting, time-based debugging, drag and drop clauses/snippets, and so forth.

- Redesigning the language, tailoring it to better express logic regarding requirement conditions, deadlines and dates, and people and their roles.

- Completing the API and creating wrapper applications that generate promise code, customized with relevant participants, dates, variables, and so forth.

Based on these possible approaches, the next iteration will likely branch the project into two. One branch will further explore the use of social commitment via natural language requests and promises, somewhat akin to Coordinator, but fast-forwarded to today’s web community bazaar context, and far less structured.
The other branch will continue development of the promise coding. While a switch to a new language and interpreter is unlikely, the promise code language may be tweaked to make it both more expressive and more natural, such as the addition of role objects into the promise (4.1.3), or additional DSL methods and object types to ease date comparison. This approach will also include better inline documentation and code search, to assist users in finding relevant code examples. The coding will also be eased simply by wrapping the entire process into external apps that use the API to generate the promise code.

5.1.2 Community & API

Another key lesson was a reminder of the difficulty of building a user base. Community-oriented web applications face a bootstrapping challenge of not being interesting or effective without enough participants, and for this very reason not being able to attract people to join. Previous PLW web projects have taken significant periods to attract enough participants to understand how the system scales.

Promiserver suffers from two issues that have limited its adoption. First, as previously noted, given the rapid development of the project there simply has not been enough time for users to find and learn the system. Second, and more importantly, other than a few nerds Promiserver is unlikely to attract people for its features in and of itself. While this thesis introduces Promiserver as a standalone application, it holds more potential as a service coupled with other systems via its API. This API allows for retrieval of XML and JSON formatted data about users, promises, and components of promises like variables and roles. More importantly, it facilitates programmatic creation and manipulation of promise data, such that other services may act as agents to generate, publish, and even sign promises and change state values.

Therefore the next step in inviting greater usage is to develop compelling applications that make use of Promiserver via the API, and that are valuable to specific communities. One such starting point could be retrofitting Promiserver into existing PLW community projects, such as OPENSTUDIO,
Kyle Buza and Takashi Okamoto’s OpenCode\(^1\), or Okamoto’s RunLog\(^2\). These task-based, data-driven community applications could benefit from a generalized, outside system for creating commitments to future action. It is this sort of integration that originally motivated Promiserver.

### 5.2 Future Work

#### 5.2.1 Dynamic Negotiation

A Promiserver author writes a promise privately, revises and debugs privately, then publishes, at which point the promise is frozen while awaiting for signatures. In contrast, most real world deals are initiated with an attempt to arrive at a “meeting of the minds,” before the first contract is even drafted. The contract will then be iteratively negotiated and revised. Only after signature is the contract frozen\(^5\). The early-freezing in Promiserver is an issue because it sidesteps this negotiation process, which is the most vital and challenging step in crafting deals. But what form will negotiation take in creating these micro-contracts?

Negotiation requires an iterative, multilateral revision process, either occurring in public or privately among members (Figure 5-1). As a first pass in approaching this topic, we may look at open content, wiki-style collaborative editing and versioning. As in these systems, participants should be able to easily and asynchronously critique, test, and contribute to the promise throughout the creation process. Promiserver is also a coding system, and contributions to promise code must be evaluated, debugged, and verified. Changes to the code must be clear, as well as how the changes will affect the promise logic.

Multi-author promises start to look a lot like open source software. Though open source development methodologies are still new, the distributed nature of the work may offer some models for development and negotiation of commitments. Yet while open content and open source projects are ostensibly

\(^1\) OpenCode - http://opencode.media.mit.edu/

\(^2\) RunLog - http://runlog.media.mit.edu/
built on individual contributions to a shared goal, contract negotiation tends to focus on finding common ground around competing individual goals. Lightweight promise systems must balance both collaborative authoring and competitive self-interest. And to stay true to the original vision of agility and dynamism, these negotiation steps must also be efficient.

### 5.2.2 Measures of Trust

As communities and markets begin to make use of trust commitment systems, we will require new approaches to facilitate browsing and summaries of participants’ promise histories. While it may at first seem tempting to simply aggregate promise history into some cumulative score, issues of promises and trust are much more nuanced than a simple rating.

Different types of commitments may vary substantially in their social or financial weight, and breaches may be remedied with varying levels of success. While social capital may eventually be a form of currency[7], it needn’t be numeric. It would be a shame to follow the path of existing simplistic credit ratings when today’s visualization and interface technologies potentiate rapid summaries and explorations of the data.
5.2.3 New Forms of Commitment

The API exposed in Promiserver only hints at the many ways in which programmatic contracts may enable forms of commitments and exchanges far beyond the scope of traditional legal contracts. Operating at the boundary between the web architecture and social system, coded promises may take inputs not only from human sources, as Promiserver works now, but from anything else on the network as well.

Rather than direct intervention by a human participant, variables could be set automatically according to news source feeds, such as stock prices or weather, or even personal data feeds, such as email. Consider the following example promise to buy stock before it hits a certain price. The aapl.price variable may be set by a news feed, while the sale.confirmation.recd variable may be triggered by receipt of a particular email.

```python
# Selling stock
if sale_confirmation_recd
    success "stock purchased"
elsif aapl_price > 91.10
    breach "stock was not purchased"
end
```

Beyond these immediate extensions, we can also foresee letting autonomous agent systems proxy for people in these micro-contracts, making decisions about which variables to set and when. For example, the stock purchase code above may be managed by a specialized buying agent. Inputs may also come from even more unusual sources, such as events in virtual worlds, or data from real world sensor values. The micro-contract code becomes a mechanism for routing inputs of data and outputs of action through social networks of commitment.

5.2.4 Networks of Promises

As a project that amalgamates and remixes ideas from a variety of fields, Promiserver offers a vision of communities as efficient networks of linked,
structured, and stateful commitments. It points the way to further research in market-based approaches to collaboration in a post-digital, post-network world. More and more, collaboration will take place not in the office or the studio, but over the net with people we do not know in person. As we move to this distributed, dynamic work model, new models of trust and commitment will be necessary.

Previous projects explored the paradox of building a virtual economy around purely digital goods, which, being infinitely copyable, quickly become valueless as they are easily redistributed for free. The value, we found, lies not in the virtual object, but in its associations to people, and in turn their associations to others. Digital market value is the result of collective action in the human network, action that emerges from individual commitments—from promises.
Appendix A

Selected Code

A.1 Constant Injection

The following code is the implementation of constant injection[24], as well as an example of it usage with Time.

# Add inject_constant to all objects.
class Module
  def inject_constant(constant, value)
    old_verbose = $VERBOSE # shut off pesky errors
    $VERBOSE = nil
    old_value = self.const_get(constant)
    begin
      self.const_set(constant, value) # Set constant to
      # the new value.
      yield if block_given?
    ensure
      const_set(constant, old_value) # Set constant back.
      $VERBOSE = old_verbose
    end
  end
end
# Add the class_shifted_to method to Time class

class Time
  class <<self
    # Creates a duplicate Time class, but with the Time.now
    # shifted to the given time.
    def class_shifted_to t
      newklass = self.dup
      newklass.instance_eval do
        def set_time t
          @_at_time = t
        end
        def now; @_at_time; end
        def new; @_at_time; end
      end
      newklass.set_time(t)
      newklass
    end
  end
end

# Example usage.
# Note: this isn’t thread safe, but we don’t run Rails
# in separate threads, just separate processes.
puts Time.now
my_new_time = Time.parse("September 1, 2005")
new_time_klass = Time.class_shifted_to(my_new_time)
Object::inject_constant(:Time, new_time_klass) do
  puts "#{Time.now} (shifted)"
end
puts Time.now

## Result
# Thu Apr 19 18:36:46 -0400 2007
# Thu Sep 01 00:00:00 -0400 2005 (shifted)
# Thu Apr 19 18:36:46 -0400 2007
A.2 Notable Promises

These are some of the early interesting promises from the first few weeks of Promiserver.

A.2.1 finish thesis

Online at http://promise.media.mit.edu/promises/1

The first promise I created was to finish this thesis, though I craftily granted myself an option to take an extension.

deadline = Time.parse("5/11/2007")
if Time.now > deadline
  if extension
    if Time.now > (deadline + extension)
      breach "Missed the extension deadline."
    end
  else
    breach "Missed the official deadline."
  end
end
success "hooray!" if completed and accepted

A.2.2 Paik’s Composition for Poor Man

Online at http://promise.media.mit.edu/promises/23

Created by Media Lab and PLW alumnus Burak Arikan, this promise stretched the system into new territory by revisiting a classic piece from the 1960’s Fluxus movement. Though clearly difficult to enforce in its current form, in combination with an open API and sensor networks, or taking place in a virtual or game world, promises like this may become possible.

# Recreation of Nam June Paik’s 1961 receipe
# original receipe:
# "Summon a taxi, position yourself inside, request a long ride, OBSERVE THE METER."
if taxi_arrived
    if in_taxi
        success "request a long ride"
    else
        breach "position yourself inside"
    end
else
    "summon a taxi"
end
success "OBSERVE THE METER" if taxi_moving

A.2.3 a little random promise

Online at http://promise.media.mit.edu/promises/30

Media Lab and PLW alumnus Carlos Rocha was the first to explore use of random outcomes. This promise waits for a certain date to pass, then essentially flips a coin to determine whether it is a breach or success. The outcome turned out to be breach.

```ruby
promise_time = Time.parse("04/25/2007")
if Time.now > promise_time
    if rand() > 0.5
        success "i did it!"
    else
        breach "this is all your fault!"
    end
end
```

98
Bibliography


http://adsense.blogspot.com/2006/02/ad-rank-explained.html.


